IBM

IBMパーソナルシステム/55™

BASIC インタープリター
ユーザーズ・ガイド
（下巻）文法編

N:SH18-2120-2

IBM

IBMパーソナルシステム/55™

BASIC インタープリター
ユーザーズ・ガイド
（下巻）文法編

N:SH18-2120-2

第 3 版　1989 年 5 月

このマニュアルは，製品の改良その他により適宜改訂されます。

© Copyright International Business Machines Corporation　1986,1989

はじめに

BASIC インタープリター　ユーザーズ・ガイドは，IBM マルチステーション 5550 で働く BASIC インタープリター バージョン K3.2 と K3.3 の使用方法や文法を詳細に説明した，参照用のマニュアルです。

このユーザーズ・ガイドは次の各編より成る 2 分冊で構成されています。

- 上巻 → 解説編
- 下巻 → 文法編

本書は**文法編**です。この文法編は，別冊の解説編と併せてお使いください。本書で「解説編」と記述しているときは，BASIC インタープリター ユーザーズ・ガイドの上巻「解説編」を指します。

解説編では，主として BASIC でできること，BASIC を使用する上での基本的な規則を説明しています。また，付録では，BASIC の持つ機能を活用するときに，有効な情報を載せています。

この**文法編**では，BASIC のコマンド，ステートメント，関数，変数をアルファベット順に説明しています。付録にはメッセージ集があります。

バージョンK3.2とK3.3の機能

BASICバージョンK3.2とK3.3で追加または拡張された機能は次のとおりです。なお，バージョンK2.0とK2.1で追加された機能については，「BASICの文法」の最後に「バージョン・アップの追加機能一覧」がありますので，これを参照してください。

バージョンK3.2の追加/拡張機能の概要

1 共用ファイルに対するアクセスの制御

- LOCK，UNLOCKステートメント
- OPENステートメント

複数の処理プログラムが入出力を行うと思われるファイルをアクセスする場合に，他のプログラムによるアクセスをどのように制御するかまたは，他のプログラムが使用しているファイルに対してどのようにオープンするかを設定できます。

2 キー処理の拡張

- ON KEY (n)，KEY (n) ステートメント
- KEYステートメント

PF11とPF12にもPF1～PF10と同じようにキー機能を割り当てることができます。また独自のキー割り込みを定義するための，キー操作に対応する番号が5つ増えています。KEY (n) は，KEY (0) と指定すると，定義されたすべての割り込みキーを制御できます。

3 その他の機能拡張

- EXTERR (n) 関数

DOS の拡張エラー情報を得ることができます。

- ERDEV 変数

  装置ヘッダーの属性ビットが追加されています。

- レコード番号の拡張

  レコード番号が 1 ～ 4,294,967,295（2^{32}-1）まで使用できます。

4　JIS83 年版文字セットの使用

DOS で JIS83 年版文字セットが指定されているときは，BASIC でも JIS83 年版文字セットが使用できます。

バージョン K3.3 について

バージョン K3.3 では，グラフィックス支援カードを使用すると，16 色カラー・グラフィックス・モードで，より高速なグラフィックス処理が可能です。各機能はバージョン K3.2 と同じです。バージョン K3.3 で高速グラフィックス処理を行うためには，必ずグラフィックス支援カードが要ります。

本書の構成

文法編

BASIC ユーザーズ・ガイド文法編は次の各部分から構成されています。

BASIC の文法

BASIC のコマンド，ステートメント，関数，変数をアルファベット順に詳しく説明しています。使用するステートメントなどの名前が分からない場合は，目的別索引（ユーザーズ・ガイドの最後に記載）またはこの「はじめに」の「機能別ステートメント一覧」を参照してください。

付録 A.メッセージ集

BASIC 使用中に画面に出力されるメッセージの意味とその対応について説明しています。

付録 B.参考資料

プログラミングする場合に役立つことがらを一覧表にして記載しています。この内容のほとんどの項目は解説編で説明している内容を簡略した表に直したものです。内容について不明な点は解説編または文法編の関連項目を参照してください。

解説編

別冊の BASIC インタープリターユーザーズ・ガイド解説編は次のように構成されています。

第 1 章 BASIC の始動と操作の方法

BASIC の始動（終了），キーの操作，プログラムの編集などについて説明しています。BASIC インタープリターは会話形式でプログラムを作成していきます。BASIC の操作に不慣れな方や，他言語の知識をお持ちの方でインタープリターは初めての方は特に確認のために参照してください。

第 2 章 プログラミングの基本規則

BASIC で使用できる文字，定数や変数における文字や数値の表現の規則，演算子，いわば BASIC 規則集です。ただし，規則だけでなく使い方につい

ても例と共に説明しています。

第 3 章 BASIC の入出力指定

この章では，BASIC でファイルや画面に対して入出力を行うときに，知っておく必要がある事柄を説明しています。画面の使い方では画面モードの機能や色の使い方を説明しています。

付録 A.ディスク入出力

BASIC では「順次ファイル」と「ランダム・ファイル」を作成することができます。この付録ではこの 2 つのファイルの作成やデータの検索を行うプログラムを記述するためには，どのような手順を踏むべきかについて例を挙げて説明しています。その他，プログラム・ファイルを扱うコマンドについても紹介しています。

付録 B.機械語サブルーチン

アセンブラーで記述したサブルーチンを，BASIC から呼び出すこともできます。その場合，記憶域の内容について配慮が必要となります。ここでは，その方法について説明しています。

付録 C.通信

BASIC で非同期通信を行う場合，ディスク・ファイルとほとんど同じようにデータの入出力を行うことができます。ただし，データをやりとりする場合のタイミングに注意を払う必要があります。ここでは，通信固有のことがらを中心に，BASIC で通信を行うためのプログラムの記述方法について説明しています。

付録 D.コード体系

BASIC で使用する文字には，すべて一連のコードが振られています。この付録では，これらのコード体系の相互関係を図で示して説明しています。BASIC ではキーの操作を操作に対応する番号で得ることもできます。また，出力装置に制御文字を送ることもできます。これらの方法は文法編と併せて参照してください。

付録 E.技術情報

記憶域マップ，変数の値の格納方法，BASIC ファイル制御ブロックなどの

技術情報を提供しています。

付録F.用語集

この用語集はBASICでプログラミングする場合に一般的に良く用いられる用語や本書で使用している特別の用語の意味を「あいうえお」順で説明しています。

本書の参照方法

BASIC インタープリターで初めてプログラミングされる方は，

解説編の第 1 章と第 2 章，第 3 章は BASIC の操作と規則を知るための説明を記述しています。すべてを覚える必要はありませんが，簡単に一読して情報の記述場所を確認してください。その後，文法編を参照してください。プログラミングの目的から使用するステートメントなどを見つけるには，ユーザーズ・ガイドの最後にある目的別索引を参照してください。解説編の付録は必要に応じて参照してください。

BASIC インタープリターによるプログラミングに慣れている方は，

この「はじめに」にある機能別ステートメント一覧または目的別索引は使用するステートメントを確認するために便利です。プログラミングのときは「BASIC の文法」を参照してください。解説編の付録は各機能を活用するために随時参照してください。

関連マニュアル

システム操作ガイド，N:GA18-2325

日本語 DOS ユーザーズ・ガイド　入門編，N:SC18-2135
日本語 DOS ユーザーズ・ガイド　参照編，N:SC18-2136
日本語 DOS の内部構造とプログラム・インターフェース，N:SC18-2128
日本語文節変換プログラム　ユーザーズ・ガイド，N:SC18-2119
ハード・ディスクをお使いいただくために，N:SC18-2004

IBM マルチステーション 5550 BASIC の操作方法，N:SR18-3785
IBM マルチステーション 5550 BASIC のプログラミング，N:SR18-3780

漢字コード一覧表，N:GC18-2040

機能別ステートメント一覧

コマンド，ステートメント，関数，変数を機能別に分けると次のようになります。ここでは予約語（BASICの命令群を記述するために決められた言葉）のみを記載しています。一部ステートメントと関数の両方に使用されている予約語がありますが両方とも関連該当個所に属するため，重複して記述していません。また，複数の機能を持つものは，一般によく使用される機能の項目に含めています。

機能概要	コマンド，ステートメント，関数，変数		
プログラムの編集	AUTO	DELETE	EDIT
	LIST	NEW	RENUM
プログラム実行の制御	CONT	CHAIN	MERGE
	RUN	SHELL	
実行順序の制御	END	FOR～NEXT	GOSUB
	GOTO	IF	ON GOSUB
	ON GOTO	RESUME	
変数の定義	COMMON	DATA	DEF型
	DIM	OPTION BASE	ERASE
	LET	READ	RESTORE
画面	CLS	CSRLIN	LOCATE
	POS	PRINT	PRINT USING
	SCREEN	VIEW PRINT	WIDTH
	WRITE		
ファイル	BLOAD	BSAVE	CLOSE
	EOF	FIELD	FILES
	GET#	PUT#	INPUT#
	INPUT$	INPUT¥	KILL
	LINE INPUT#	LOAD	LOC

	LOF	LSET, RSET	NAME
	OPEN	PRINT#	PRINT# USING
	RESET	SAVE	WRITE#
	LOCK	UNLOCK	
ディレクトリー	CHDIR	MKDIR	RMDIR
キーボード	INKEY$	INPUT	INPUT$
	INPUT¥	KEY (n)	KEY
	LINE INPUT	ON KEY	
プリンター	LLIST	LPOS	LPRINT
	LPRINT USING	WIDTH	
通信	COM (n)	ON COM (n)	OPEN "COM
	WIDTH		
図形処理	CIRCLE	DRAW	GET
	PUT	LINE	PAINT
	PMAP	POINT	PRESET
	PSET	VIEW	WINDOW
色	COLOR	PALETTE	
	PALETTE USING		
音	ON PLAY (n)	PLAY	PLAY (n)
	SOUND		
記憶域	CALL	CLEAR	DEF SEG
	DEF USR	FRE	PEEK
	POKE	USR	VARPTR
	VARPTR$		
ポート	INP	OUT	WAIT

日時 / 時間	ON TIMER (n)	DATE$	TIME$
	TIMER		
エラー	ON ERROR	ERR, ERL	ERDEV
	ERDEV$	EXTERR	
ストリングの操作	CDBL$	CSNG$	INSTR
	KPOS	LEFT$	LEN
	MID$	RIGHT$	SPACE$
	STRING$		
文字とコード	ASC	CHR$	JIS$
	KTN$		
数学関数	ABS	ATN	COS
	EXP	INT	LOG
	SIN	SQR	TAN
数値精度	CDBL	CINT	CSNG
	FIX	SGN	
ストリング / 数値変換	CVI, CVS, CVD	HEX$	
	MKI$, MKS$, MKD$		OCT$
	SPC	STR$	TAB
	VAL		
乱数	RANDOMIZE	RND	
デバッグ	TRON, TROFF	REM	
環境	ENVIRON	ENVIRON$	
装置駆動ルーチン	IOCTL	IOCTL$	
BASICの終了	SYSTEM		

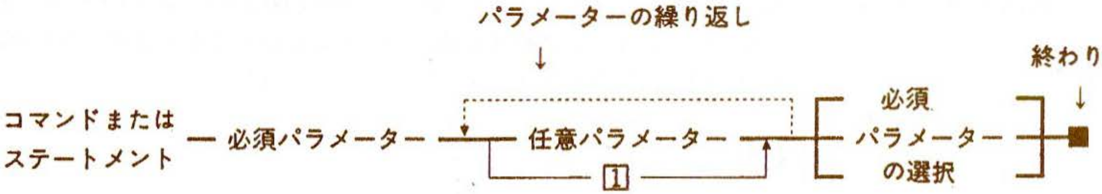
構文図の見方

構文図は次の規則に従って読んでください。

- 線に沿って左から右に読む。
- 線の分岐点ではいずれか任意の線を選択する。
- 矢印の方向に沿って読み進む。

構文図は各パラメーターの機能を次のように表しています。

コマンド、ステートメント



コマンドまたはステートメント　各コマンドやステートメントを識別するための名前を指定します。これはBASICの予約語で，コマンドやステートメントの始まりを表すものです。

【例】

影の部分が該当項目です。

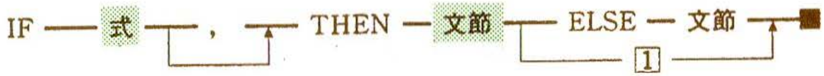
必須パラメーター

省略できないパラメーターです。構文図では省略が可能であることを示す分岐がありません。

【例】

```
20 IF A=D THEN GOTO 100
```

この IF ステートメントは構文図で次のように記述しています。次の影の部分が必須パラメーターです。



1 式が偽のときに実行される文節なし。

任意パラメーター

省略可能なパラメーターです。図中の四角で囲まれた数字は省略値があることを表しています。省略値は構文図の下に記述してあります。次の例の影の部分（行）は任意パラメーターです。

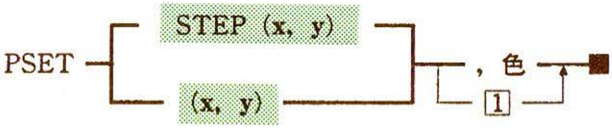
【例】

1 最小の行番号

必須パラメーターの選択

構文図に指定されたパラメーターのうちいずれか1つを必ず選択するパラメーターです。次の例の影の部分は必須パラメーターです。

【例】

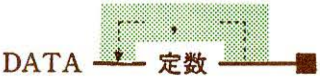


[1] 前景色

パラメーターの繰り返し

点線で分岐しているパラメーターは複数指定できるパラメーターです。点線の中にあるカンマ（,），コロン（:），セミコロン（;）はパラメーターの区切り記号です。

【例】



この構文図はパラメーター「定数」をカンマ（,）で区切って複数指定できることを表しています。

終わり

コマンドまたはステートメントの終わりです。構文図では中塗りの四角で表しています。

【例】

関数

―― 関数名（引数）――

関数名　BASIC の組み込み関数名で，BASIC の予約語です。

引数　関数に渡す引数です。

変数

―― 変数名 ――

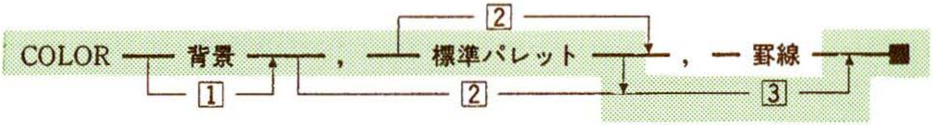
変数名　BASIC に組み込まれた変数で，予約語です。

読み方の例

COLOR ステートメントを例にとって指定可能な形式のうち 2 とおりを構文図と実際のステートメントで表します。構文図の見方とステートメントの記述を確認してください。

1　「**背景**」に「**3**」を，「**罫線**」に「**1**」を指定するとき。

構文図は次の影の部分のように読みます。



1 2 3 前の値

＜プログラム中のステートメント＞

```
20 COLOR 3,,1
```

2　「**標準パレット**」に「**1**」を指定するとき。

構文図は次の影の部分のように読みます。

1 2 3 前の値

＜プログラム中のステートメント＞

```
20 COLOR ,1
```

文法編の目次

BASICの文法

ABS ………… 絶対値 ………… 1
ASC ………… 連続コード ………… 2
ATN ………… 逆正接 ………… 3
AUTO ………… 行番号の自動作成 ………… 4
BEEP ………… スピーカーを鳴らす ………… 6
BLOAD ………… 記憶域イメージ・ファイルの読み込み ………… 7
BSAVE ………… 記憶域イメージ・ファイルの保管 ………… 9
CALL ………… 機械語サブルーチンの呼び出し ………… 10
CDBL ………… 倍精度に変換 ………… 11
CDBL$ ………… 1バイト・コード英数字を2バイト・コードに ………… 12
CHAIN ………… 他プログラムに制御を移す ………… 13
CHDIR ………… 現行ディレクトリーの変更 ………… 16
CHR$ ………… 連続コードを文字に変換 ………… 18
CINT ………… 整数に変換 ………… 21
CIRCLE ………… 円を描く ………… 22
CLEAR ………… データの消去 ………… 26
CLOSE ………… ファイルのクローズ ………… 28
CLS ………… 画面の消去 ………… 30
COLOR ………… 文字の色と属性，図形や背景の色指定 ………… 32
COM (n) ………… 通信割り込み処理の制御 ………… 42
COMMON ………… 連鎖されたプログラムに制御を渡す ………… 43
CONT ………… 実行の再開 ………… 44
COS ………… 余弦 ………… 46
CSNG ………… 単精度に変換 ………… 47
CSNG$ ………… 2バイト・コード英数字を1バイト・コードに ………… 48
CSRLIN ………… 現在のカーソルの行位置 ………… 49
CVI, CVS, CVD ………… ストリング型から数値型に変換 ………… 50
DATA ………… 数値定数やストリング定数の記憶 ………… 52

DATE$ …… 日付の読み取りと設定 …… 53
DEF FN …… ユーザー関数の定義 …… 55
DEF SEG …… セグメントの定義 …… 57
DEF 型 …… 変数の型を宣言する …… 58
DEF USR …… 機械語サブルーチンの開始アドレスの指定 …… 60
DELETE …… 行の削除 …… 61
DIM …… 配列の宣言 …… 62
DRAW …… 線を描く …… 64
EDIT …… 行の表示と編集 …… 69
END …… 実行の終了 …… 70
ENVIRON …… 環境にパラメーターを設定する …… 71
ENVIRON$ …… 環境から値を読み取る …… 73
EOF …… ファイルの終了状態を返す …… 75
ERASE …… 配列の消去 …… 76
ERDEV, ERDEV$ …… 装置のエラー情報 …… 77
ERR, ERL …… エラー・コードと行番号を得る …… 79
ERROR …… エラー発生の模擬とエラー・コード定義 …… 81
EXP …… 指数関数 …… 83
EXTERR (n) …… 拡張エラー情報を得る …… 84
FIELD …… レコードの項目定義 …… 86
FILES …… ファイル名の表示 …… 89
FIX …… 整数変換 …… 91
FOR, NEXT …… 処理の繰り返し …… 92
FRE …… 記憶域の未使用空間のバイト数を得る …… 96
GET (ファイル) …… レコードの読み込み …… 97
GET (グラフィック) …… 複数の点の読み取り …… 99
GOSUB …… サブルーチンに行き，戻る …… 101
GOTO …… 分岐 …… 103
HEX$ …… 10 進数を 16 進数に変換 …… 104
IF …… 判断 …… 105
INKEY$ …… キーボードから文字を読み取る …… 108
INP …… 入力ポートの 1 バイトを返す …… 110
INPUT …… キーボード入力 …… 111
INPUT# …… 順次ファイルから変数に読み取る …… 114

INPUT$……………………… ファイルから文字を読み取る ……………………… 116
INPUT¥ ……………………… ファイルから文字を読み取る ……………………… 118
INSTR ……………………… ストリング内の探索 ……………………… 119
INT ……………………… 整数変換 ……………………… 121
IOCTL ……………………… 装置駆動ルーチンへ制御データを渡す ……………………… 122
IOCTL$……………………… 装置駆動制御データ実行確認 ……………………… 123
JIS$ ……………………… JIS16 進コードを返す ……………………… 124
KEY……………………… PF キー表示，割り込み処理キーの設定 ……………………… 125
KEY (n) ……………………… キーの割り込み処置の制御 ……………………… 130
KILL ……………………… ファイルの削除 ……………………… 132
KLEN ……………………… 文字数を返す ……………………… 133
KPOS ……………………… ストリング内の文字位置を返す ……………………… 134
KTN$ ……………………… JIS 区点コードを返す ……………………… 135
LEFT$ ……………………… 左から指定バイト数の文字を返す ……………………… 136
LEN ……………………… 文字のバイト数を返す ……………………… 138
LET ……………………… 式の値を変数に代入する ……………………… 139
LINE ……………………… 線，矩形を描く ……………………… 140
LINE INPUT……………………… キーボードから 1 行読み込む ……………………… 144
LINE INPUT#……………………… 順次ファイルから 1 行読み込む ……………………… 145
LIST ……………………… プログラムの表示 ……………………… 147
LLIST……………………… プリンター出力 ……………………… 149
LOAD……………………… プログラムの読み込み ……………………… 150
LOC ……………………… ファイル内の現在位置を得る ……………………… 152
LOCATE ……………………… カーソル位置とカーソル表示形態の指定 ……………………… 153
LOCK ……………………… レコードの保護指定 ……………………… 156
LOF ……………………… ファイルの長さを返す ……………………… 157
LOG……………………… 自然対数 ……………………… 158
LPOS ……………………… プリント・ヘッドの現在位置を得る ……………………… 159
LPRINT……………………… プリンター印刷 ……………………… 160
LSET, RSET……………………… ストリングの左寄せ，右寄せ ……………………… 162
MERGE ……………………… プログラムのマージ ……………………… 163
MID$ ……………………… ストリングの置き換え ……………………… 164
MID$ ……………………… ストリングの一部を返す ……………………… 166
MKDIR ……………………… ディレクトリーの作成 ……………………… 168

MKI$, MKS$, MKD$	数値からストリングに変換	169
NAME	ファイル名の変更	170
NEW	記憶域内のプログラムや変数の消去	171
NEXT	ループを閉じる	172
OCT$	10 進数を 8 進数に変換	173
ON COM (n)	通信割り込み処理ルーチンの指定	174
ON ERROR	エラー発生時処理ルーチンの指定	176
ON GOSUB (GOTO)	サブルーチンまたは行への分岐	177
ON KEY (n)	キー入力処理ルーチンの指定	178
ON PLAY	音が一定以下の時の処理ルーチンの指定	180
ON TIMER (n)	時間経過時処理ルーチンの指定	183
OPEN	ファイルのオープン	185
OPEN"COM…	通信ファイルのオープン	192
OPTION BASE	配列添字の最小値指定	198
OUT	出力ポートへ 1 バイト送る	199
PAINT	色や模様を塗る	200
PALETTE	パレットに色を設定する	206
PEEK	記憶域から 1 バイト読み取る	208
PLAY	音楽演奏	209
PLAY (n)	音楽バッファーの音の数を返す	214
PMAP	ウインドーとビューポート間の座標変換	215
POINT	点の色を返す	216
POKE	記憶域に 1 バイト書き込む	218
POS	カーソルの桁位置を返す	219
PRESET	点を描く	220
PRINT	画面にデータを表示する	221
PRINT USING	指定された書式で画面にデータを表示	224
PRINT#	データをファイルへ書き込む	229
PSET	点を描く	232
PUT (ファイル)	レコードの書き込み	234
PUT (グラフィック)	図形を画面に設定する	235
RANDOMIZE	乱数シードの再設定	241
READ	DATA ステートメントから値を読み取る	243
REM	注釈行の指定	245

RENUM …… 行番号の変更 …… 246
RESET …… ファイルのクローズとバッファーの消去 …… 248
RESTORE …… DATA ステートメントの行指定 …… 249
RESUME …… 再実行 …… 250
RETURN …… サブルーチンから戻る …… 252
RIGHT$ …… ストリングの右から任意バイトの文字を返す …… 253
RMDIR …… ディレクトリーの消去 …… 255
RND …… 乱数発生 …… 257
RSET …… ストリングの右よせ …… 259
RUN …… プログラムの実行 …… 260
SAVE …… プログラムの保管 …… 262
SCREEN …… 文字の属性を返す …… 264
SCREEN …… 画面モードの設定 …… 267
SGN …… 符号を返す …… 270
SHELL …… DOS コマンドの実行 …… 271
SIN …… 正弦 …… 273
SOUND …… 音の発生 …… 274
SPACE$ …… ストリングのスペース指定 …… 277
SPC …… スペースの設定(PRINT と共に) …… 278
SQR …… 平方根 …… 279
STOP …… 実行の中断 …… 280
STR$ …… 数式をストリングに …… 281
STRING$ …… 文字を指定個数並べてストリングに …… 282
SWAP …… 変数の値の交換 …… 283
SYSTEM …… DOS に戻る …… 284
TAB …… スペースの挿入(PRINT と共に) …… 285
TAN …… 正接 …… 286
TIME$ 変数, …… 時刻の読み取りと設定 …… 287
TIMER …… 現在の時刻を秒単位で返す …… 289
TIMER …… 経過時間ごとの割り込み処理を制御する …… 290
TRON, TROFF …… トレースの開始と中止 …… 291
UNLOCK …… レコード保護指定の解除 …… 292
USR …… 機械語サブルーチンの呼び出し …… 294
VAL …… ストリングから数値に …… 295

VARPTR …… 変数，ファイル制御ブロックのアドレス …… 298
VARPTR$ …… 変数のアドレスをストリングで返す …… 300
VIEW …… ビューポートの定義 …… 301
VIEW PRINT …… 文字表示領域の定義 …… 305
WAIT …… 実行の保留 …… 306
WHILE，WEND …… 条件が真のときにループ内を実行 …… 308
WIDTH …… 出力行の幅 …… 310
WINDOW …… ウインドーの定義 …… 313
WRITE …… データの画面表示(区切り文字つき) …… 317
WRITE# …… 順次ファイルへの書き込み（区切り文字つき） …… 318

付録A メッセージ集

付録B 参照資料

変数型宣言文字 …… B-1
演算子 …… B-1
ファイル指定とパス名 …… B-3
　ファイル指定 …… B-3
　パス名 …… B-3
装置名とドライブ指定子 …… B-3
　装置名 …… B-3
　ドライブ指定子 …… B-3
画面モード …… B-4
数学関数の公式 …… B-6
キーボードと走査コード …… B-8
数値変換表 …… B-9

解説編の目次

第1章 BASICの始動と操作の方法

BASICの始動 …… 1-1
日付と時刻の入力 …… 1-1
DOSプロンプト …… 1-1
BASICコマンド …… 1-1
DOSに戻る …… 1-1
システム・リセット …… 1-2
ハード・ディスク …… 1-2
キー操作 …… 1-3
文字を入力するキー …… 1-4
英数字の入力 …… 1-4
カタカナの入力 …… 1-5
ひらがなの入力 …… 1-5
漢字の入力 …… 1-5
数値の入力 …… 1-10
プログラムを編集するキー …… 1-11
プログラムを入力する …… 1-11
カーソル移動キー …… 1-11
編集キー …… 1-12
実行を制御するキー …… 1-13
予約語を表示するキー …… 1-13
印刷の指定を行うキー …… 1-14
BASICのプログラム …… 1-15
行番号 …… 1-16
ステートメント …… 1-16
注釈 …… 1-17
プログラムの作成と実行 …… 1-19
プログラムの作成 …… 1-19
プログラムの実行 …… 1-19

プログラムの保管 …… 1-19
プログラムの読み込み …… 1-19
リストの表示 …… 1-20
プログラム実行のモード …… 1-21
直接モード …… 1-21
間接モード …… 1-21
プログラムの修正 …… 1-22
文字の変更 …… 1-23
文字の削除 …… 1-23
文字の挿入 …… 1-24
行の消去 …… 1-24
行の追加 …… 1-24
行の削除 …… 1-26
プログラムの削除 …… 1-26
構文エラー …… 1-27
変数の内容を調べる …… 1-28
BASIC 始動時に指定できる項目 …… 1-29

第2章 プログラミングの基本規則

BASIC で使用できる文字 …… 2-1
特殊文字 …… 2-2
予約語 …… 2-3
BASIC で使用できるデータ …… 2-6
定数 …… 2-7
ストリング定数 …… 2-7
数値定数 …… 2-7
整数 …… 2-7
固定小数点数 …… 2-7
浮動小数点数 …… 2-8
16 進数 …… 2-8
8 進数 …… 2-9
JIS16 進数 …… 2-9
JIS 区点定数 …… 2-9

変数 ………… 2-10
　変数名 ………… 2-11
　変数型 ………… 2-11
数値精度 ………… 2-13
　数値精度の変換 ………… 2-15
配列 ………… 2-18
数式の演算子 ………… 2-21
算術演算子 ………… 2-22
　整数除算 ………… 2-22
　剰余の計算 ………… 2-23
関数演算子 ………… 2-24
　数値比較 ………… 2-24
　ストリング比較 ………… 2-25
論理演算子 ………… 2-26
　論理演算子の働き ………… 2-28
実行順序 ………… 2-31
ストリングの演算子 ………… 2-33
連結演算子 ………… 2-34

第3章 BASICの入出力指定

ファイル ………… 3-1
　ファイル番号 ………… 3-2
ファイル指定とパス ………… 3-2
　ファイル指定 ………… 3-2
　パス名 ………… 3-3
　装置名 ………… 3-4
　ファイル名 ………… 3-5
ディレクトリー ………… 3-6
　ディレクトリーの構造 ………… 3-6
ディレクトリーのタイプ ………… 3-7
　現行ディレクトリー ………… 3-7
画面の使用 ………… 3-8

画面モード …… 3-9
単色システム …… 3-10
カラー・システム …… 3-11
文字モード …… 3-12
グラフィック・モード …… 3-14
座標指定 …… 3-14
ウインドーとビューポート …… 3-16
色 …… 3-21
色の使用と画面モード …… 3-26
単色文字モード …… 3-26
カラー文字モード …… 3-27
単色グラフィック・モード …… 3-28
4 色カラー・グラフィック・モード …… 3-29
16 色カラー・グラフィック・モード …… 3-30
画面表示 …… 3-33

付録A ディスク入出力

プログラム・ファイルの管理 …… A-1
プログラムの保管 …… A-1
プログラムの読み込み …… A-2
プログラムの実行 …… A-2
重ね合わせ …… A-3
ファイルの削除 …… A-4
ファイル名の変更 …… A-4
順次ファイル …… A-5
順次ファイルの書き込み …… A-7
順次ファイルの読み取り …… A-9
順次ファイルのプログラム例 …… A-10
プログラムの目的 …… A-10
出力例 …… A-10
プログラム・リスト 1 …… A-11
プログラム・リスト 2 …… A-12
ランダム・ファイル …… A-15
ランダム・ファイルの書き込み …… A-18
ランダム・ファイルの読み取り …… A-20

ランダム・ファイルのプログラム例 …… A-22
プログラムの目的 …… A-22
プログラム・リスト …… A-23

付録B 機械語サブルーチン

機械語サブルーチンの処理概略 …… B-1
サブルーチンを記憶域に置く …… B-2
なぜ整数配列なのか …… B-2
サブルーチン・ファイル …… B-2
小さいサブルーチン …… B-3
READで …… B-4
POKEで …… B-6
サブルーチン・ファイル 1 …… B-8
値の受け渡し 1 …… B-10
値の受け渡し 2 …… B-15
サブルーチンの呼び出し …… B-18
CALL ステートメント …… B-20
USR 関数 …… B-22

付録C 通信

通信ファイルの入出力 …… C-1
送信停止と送信開始 …… C-2
入出力関数 …… C-2
入力ステートメント …… C-3
通信端末 …… C-4
ファイルの送受信 …… C-6
プログラミングの注意 …… C-8
半2重と全2重 …… C-8
交換回線と専用回線 …… C-8
かな/漢字の通信 …… C-8
雑音対策 …… C-9
制御信号の操作 …… C-10
OPEN 時の出力信号 …… C-10

入力制御信号 …… C-10
モデム装置制御信号 …… C-11
出力制御信号 …… C-13
通信エラー …… C-15
ファイルのオープン時 …… C-15
データの読み取り時 …… C-15
データの書き込み時 …… C-15

付録D コード体系

コード体系 …… D-1
連続コード（10 進数） …… D-1
内部コード（16 進数） …… D-1
JIS 区点コード（10 進数） …… D-1
JIS16 進コード（16 進数） …… D-1
1 バイト・コード文字セット …… D-3
制御文字と連続コード …… D-8
コード間の変換 …… D-10
2 バイト・コード文字セット …… D-11
連続コードと内部コードの関係 …… D-12
JIS 区点コードと内部コードの関係 …… D-13
JIS16 進コードと内部コードの関係 …… D-15
コード間の変換 …… D-16
拡張コード …… D-19

付録E 技術情報

記憶域マップ …… E-1
変数の記憶方法 …… E-3
ストリング変数の記憶 …… E-3
数値変数の記憶 …… E-4
BASIC のファイル制御ブロック …… E-5
キーボード・バッファー …… E-7
印刷制御文字の指定 …… E-8
プログラム例 1 …… E-10
プログラム例 2 …… E-11

付録F グラフィックス

BASIC のグラフィックス …… F-1
円グラフ …… F-2
　プログラムの目的 …… F-2
　出力例 …… F-2
　プログラム・リスト …… F-3
プログラムの処理内容 …… F-7
　システムを調べる …… F-7
　変数の初期設定 …… F-8
　データの入力 …… F-8
　百分率を求める …… F-9
　角度を求める …… F-9
　円グラフを描く …… F-9
　円グラフを塗る …… F-9
　項目名を表示する …… F-9
　項目名と百分率を表示 …… -10
　プログラムの終り …… F-10
棒グラフ …… F-11
　プログラムの目的 …… F-11
　出力例 …… F-11
　プログラム・リスト …… F-12
プログラムの処理内容 …… F-15
　軸を描く …… F-15
　データの入力 …… F-16
　棒を描く …… F-16
絵グラフ …… F-17
　プログラムの目的 …… F-17
　出力例 …… F-17
　プログラム・リスト …… F-18
プログラムの処理内容 …… F-21
　車を表示するサブルーチン …… F-22
　値に従って車を表示 …… F-22

付録G 用語集

BASICの文法

ABS 関数

目的：

数式の絶対値を返します。

構文：

── ABS (**x**) ──

説明：

x には，数式を指定します。この関数から返される値は，常にゼロ以上の数です。

例：

```
PRINT ABS(7*(-5))
 35
Ok
```

7＊（-5）の値は-35 となり，その絶対値 35 が返されます。

ASC 関数

目的：　ストリングの最初の文字の連続コードを返します。

構文：　── ASC (**x$**) ──

説明：　x$ には，ストリング式を指定します。

この関数から返される値は，ストリングの最初の文字の連続コードを示す数値です（連続コードについては，☞解説編　付録 D「コード体系」）。

x$ が空ストリングの場合，つまり x$ として何も指定しない場合 Illegal function call（違法関数呼び出し）エラーとなります。

連続コードを文字に変換するためには，CHR$ 関数を使用します。

例：　次のプログラム例は，最初に ASC 関数を使って「漢字」というストリングから「漢」の連続コード（2074）を求め，次に CHR$ 関数を使って 2074 から文字（「漢」）を求め，これらを表示しています。

```
10 '--- ASCとCHR$ ---
20 Y$="漢字"
30 A=ASC(Y$)    '--- 文字から連続コードに変換
40 B$=CHR$(A)   '--- 連続コードから文字に変換
50 PRINT Y$,A,B$
60 END
RUN
漢字           2074          漢
Ok
```

ATN 関数

目的：

数値の逆正接を返します。

構文：

—— ATN (x) ——

説明：

x には，数式を指定します。

この関数から返される数値は，その正接が x となる角度をラジアンで表したもので，－ π 2 ～ π 2 の範囲の値です（ただし，π=3.141593 として計算されます）。

関数の返した角度をラジアンから度に変換するためには，プログラムで 180/3.141593 を掛ける必要があります。

この関数は，単精度で計算した値を返します。BASIC を始動するときに/D を指定すると倍精度で計算した値を返します。

例：

1

```
PRINT ATN(3)
 1.249046
Ok
```

2

```
10 PI=3.141593
20 RADIANS=ATN(1)
30 DEGREES=RADIANS*180/PI
40 PRINT RADIANS,DEGREES
RUN
 .7853983      45
Ok
```

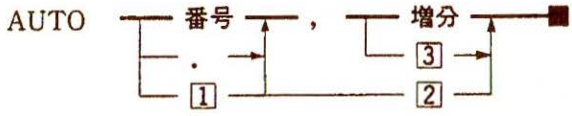
1.のプログラムは，3 の逆正接をラジアンで求めています。**2.**のプログラムは，1 の逆正接をラジアンで求め（.7853983 ラジアン），さらに度に変換しています（45 度）。

AUTO コマンド

目的：

[↵] キーを押すたびに，自動的に行番号を表示します。

構文：



[1] 10（増分が指定されたときは 0）
[2] 10
[3] 前の値

説明：

番号は，最初の行番号です。**ピリオド（.）**を指定すると，現在行の次の行番号から番号付けが開始されます。**番号**も**ピリオド**も省略すると，**増分**を指定しない場合は 10，**増分**を指定する場合は 0 が，それぞれ最初の行番号として用いられます。

増分は，新しい行番号を作成するために，直前の行番号に加えられる値です。つまり，[↵] キーを押すたびに，**増分**ずつ増加した行番号が作成されます。**カンマ（,）**も**増分**も省略すると，省略時の値として 10 が用いられます。また，**カンマ**を記述して**増分**を省略すると，前回の AUTO コマンドで使用した値が用いられます。

行番号の直後にアステリスク（*）が表示された場合，この番号がすでにプログラム内に存在することを示します。番号に続けて何らかの文字を入力すると，プログラム内の既存の行が置き換えられます。既存の行を置き換えない場合は，アステリスクの直後に [↵] キーを押し，次の行番号を表示させます。

AUTO の番号付けを終了させるためには，[Break] キー機能を押します。[Break] キー機能を押した時点の行は保存されず，コマンド入力のレベルに戻ります。

例：

1 `AUTO`

行番号 10,20,30,40,...を作成します。

2 `AUTO 100,50`

行番号 100,150,200,250,...を作成します。

3 `AUTO 500,`

前回の AUTO コマンドの増分を 50 とすると，行番号 500,550,600,650,...を作成します。

4 `AUTO ,20`

行番号 0,20,40,60,...を作成します。

BEEP ステートメント

目的：　スピーカーを鳴らします。

構文：　BEEP ──■

説明：　スピーカーから，800Hz の音が 1/4 秒間鳴ります（これをビープ音と呼びます）。これは，次のステートメントと同様の機能です。

```
PRINT  CHR$(7);
```

BLOAD コマンド

目的：

記憶域イメージのファイル（機械語プログラムなど）を，記憶域に読み込みます。

構文：

BLOAD ― **ファイル指定** ―┬― **，オフセット** ―┬―■
　　　　　　　　　　　　　└――― [1] ―――┘

[1] BSAVE で指定されたオフセット

説明：

ファイル指定には，「装置名：」または「ドライブ指定子：ファイル名」形式のストリング式で，ファイルを指定します（ファイル名については，☞解説編　第 3 章「ファイル指定とパス」)。

装置名またはドライブ指定子を省略すると，省略時ドライブの現行ディレクトリーが参照されます。また，ドライブ指定子に続けてパス名を指定することもできます。

ファイル指定に間違いがある場合，Bad file name（ファイル名の誤り）エラーとなり，読み込みは取り消されます。

オフセットは，0 ～ 65535 の範囲の数式で指定します。この値は，直前の DEF SEG で宣言された記憶域のセグメント内のオフセットを示し，ファイルはこのアドレスから始まる記憶域に読み込まれます。

オフセットを省略すると，このファイルを BSAVE で保管した際のオフセット値が使用されます（つまり，保管前と同じ位置に読み込まれます)。

BASIC は，アドレスが正しいかどうかの検査は行いません。したがって，**オフセット**を指定する際，読み込み先のアドレスに注意する必要があります。

BLOAD コマンドは，BASIC プログラムから他の機械語サブルーチンを実行する場合に役立ちます。BLOAD で機械語サブルーチンを読み込み，BASIC プログラム内の CALL ステートメントで実行させます。

機械語サブルーチンの読み込みについては，☞ 解説編，付録 B「機械語サブルーチン」を参照。

BSAVE コマンド

目的：　記憶域の一部分のイメージを，指定した装置に保管します。

構文：　BSAVE ── **ファイル指定，オフセット，長さ** ──■

説明：　**ファイル指定**には，「装置名：」または「ドライブ指定子：ファイル名」形式のストリング式で，ファイルを指定します（ファイル名については，☞解説編　第3章「ファイル指定とパス」）。

装置名またはドライブ指定子を省略すると，省略時ドライブの現行ディレクトリーが参照されます。また，ドライブ指定子に続けてパス名を指定することもできます。

ファイル指定に間違いがある場合，Bad file name（ファイル名の誤り）エラーとなり，保管は取り消されます。

オフセットは，0～65535の範囲の数式で指定します。この値は，直前のDEF SEGで宣言されたセグメント内のオフセットを示し，このアドレスから始まる記憶域のイメージが保管されます。

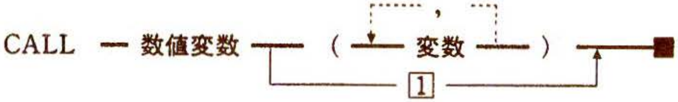
長さは，1～65535の範囲の数式で指定します。これは，保管する部分のバイト数を示します。

オフセットや**長さ**を省略すると，Syntax error（構文エラー）となり，保管が取り消されます。また，BASICはアドレスが正しいかどうかを検査しないため，**オフセット**を指定する際には,アドレスの指定に注意が必要です。

CALL ステートメント

目的：　機械語サブルーチンを呼び出します。

構文：



1　機械語サブルーチンに渡す引数は無し

説明：

数値変数には，数値変数名を指定します。この変数の値をオフセットとするアドレス（直前の DEF SEG で定義された記憶域のセグメント内）から，機械語サブルーチンが呼び出されます。したがって，CALL ステートメントの使用前に，サブルーチンの開始位置を示すオフセットの値を変数に与えておきます。

BASIC はアドレスが正しいかどうかを検査しないため，記憶域の範囲外のアドレスをアクセスしないよう，**数値変数**の値に注意してください。

変数は，引数として機械語サブルーチンに渡される変数の名前です。これはカンマ（,）で区切って任意の数だけ指定することができます。

機械語プログラムと BASIC プログラムのインターフェースをとる方法として，他に USR 関数があります（USR 関数を参照）。

機械語サブルーチン使用上の注意事項は，☞　解説編　付録 B「機械語サブルーチン」

CDBL 関数

目的：

数式を倍精度の数値に変換します。

構文：

—— CDBL (**x**) ——

説明：

x には，数式を指定します。

数値の精度変換の規則については， 解説編　第 2 章の「数値精度の変換」。

返される数値の下位の桁については，x で与えられた有効数字以降の数字は意味の無い数字です。

数値を整数に変換する CINT 関数，および単精度に変換する CSNG 関数の記述も参考にしてください。

例：

```
10 A=454.67
20 PRINT A;CDBL(A)
RUN
 454.67  454.6700134277344
Ok
```

この例では，454.67 を倍精度に変換しています。したがって，倍精度の数値の有効数字は，小数点以下 2 桁までとなります。

CDBL$ 関数

目的：

ストリング内の 1 バイト・コードの英数字を，すべて 2 バイト・コードに変換します。

構文：

—— CDBL$ (x$) ——

説明：

x$ には，ストリング式を指定します。

変換されるものは，1 バイト・コード（連続コード 255 以下）の数字，英小文字，および英大文字であり，それぞれ 2 バイト・コード（連続コード 256 以上）の数字，英小文字，および英大文字になります。これ以外の文字は変換されません。

ストリング式の長さは，結果が 255 バイト以内になる範囲でなければなりません。コード変換の結果が 255 バイトを超える場合，String too long（ストリングが長すぎる）エラーが返されます。

2 バイト・コードの英数字を 1 バイト・コードに変換するためには，CSNG$ 関数を使用します。

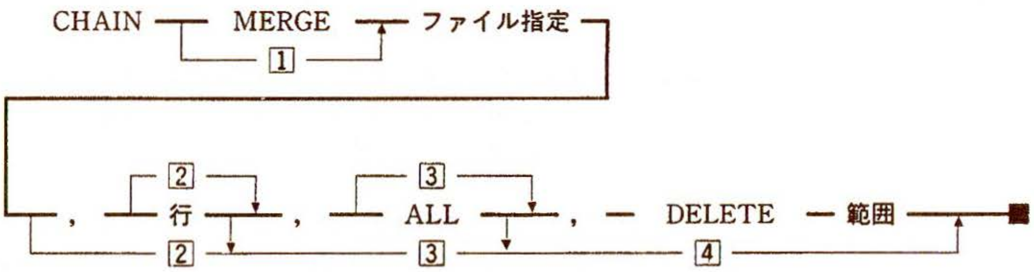
例：

```
10 PRINT CDBL$("1234567890")
20 PRINT CDBL$("+3.141592654")
30 PRINT CDBL$("-1000")
40 PRINT CDBL$("California")
RUN
１２３４５６７８９０
＋３．１４１５９２６５４
－１０００
Ｃａｌｉｆｏｒｎｉａ
Ok
```

CHAIN ステートメント

目的： 他のプログラムを読み込み，実行の制御を移します（これを連鎖と呼びます）。

構文：



1 マージしない
2 連鎖プログラムの最初の行
3 COMMON ステートメントで宣言した変数以外は渡らない
4 現プログラムの行を削除しない

説明： **ファイル指定**には，「装置名：」または「ドライブ指定子：ファイル名」形式のストリング式で，連鎖するプログラムのファイルを指定します（ファイル名については，☞ 解説編　第 3 章「ファイル指定とパス」）。

装置名またはドライブ指定子を省略すると，省略時ドライブの現行ディレクトリーが参照されます。ドライブ指定子に続けてパス名を指定することもできます。

ファイル指定に間違いがある場合，File not found（ファイル名の誤り）エラーが起こり，このステートメントは無効になります。

行には，数式を指定します。この値は，連鎖プログラム内の実行を開始する行番号を示します。**行**を省略すると，連鎖プログラムの最初から実行されます。

連鎖プログラムの行番号をRENUMコマンドで付け直した場合，CHAINの行の指定を修正する必要があります。

MERGEは，現プログラムに連鎖プログラムを組み込む指定です。現プログラムで宣言された変数型の設定は，組み込まれた連鎖プログラムにも有効です。MERGEを指定する場合，連鎖プログラムは文字形式ファイル（SAVEコマンドで，Aを指定）である必要があります。

MERGEを省略すると，現プログラムの変数型は，連鎖プログラムでは無効です。したがって，現プログラムと同じ変数型を使用する場合，連鎖プログラムで再び宣言する必要があります。

ALLは，現プログラムの全変数およびOPTION BASEの設定を連鎖プログラムに渡す指定です。ALLを省略すると，現プログラムのCOMMONステートメントで宣言された変数以外は，連鎖プログラムに渡りません（COMMONステートメント参照）。

連鎖プログラムでは，ON ERRORなどのON…ステートメントやユーザー定義関数は無効です。

DELETEは，連鎖プログラムを組み込む前に，一定範囲の現プログラム行を削除します。これにより，連鎖プログラムの開始行番号から終了行番号の範囲に該当する現プログラム行を削除しておき，連鎖プログラム内に余分な行が入ることを防ぐことができます。DELETEは，**範囲**と共に指定する必要があります。

DELETEを省略すると，連鎖プログラムの開始行番号から終了行番号の範囲に該当する現プログラム行がある場合，同じ行番号の連鎖プログラム行で置き換えられない限り，この行は連鎖プログラムの一部として実行されます。

範囲には，削除する部分の最初と最後の行番号をハイフン（-）で結んで指定します。現プログラムの行番号をRENUMコマンドで付け直した場合，**範囲**に指定した2つの行番号も自動的に修正されます。

ファイルは，CHAIN ステートメントの実行による影響を受けません。現プログラムでオープンされたファイルは，そのままの状態で連鎖プログラムに引き継がれます。

READ によるデータのアクセスは，CHAIN の実行と同時に初期状態に戻ります。つまり，連鎖プログラム内の READ ステートメントは，最初の DATA ステートメントの 1 番目の項目からアクセスします（これは，RESTORE ステートメントの効果と同様です)。

C

例：

1
```
CHAIN "A:PROG1"
```

この例では，ファイル指定以外の項目を省略しているため，連鎖プログラムは，現プログラムの変数型宣言などによる影響を受けません。また，特に指定のない変数は渡されません。連鎖プログラムは最初の行から実行されます。

2
```
CHAIN "A:PROG1",1000
```

この例では，ファイル指定と行を指定しています。1.と同様に，現プログラムの影響は受けず，指定のない変数も渡されません。連鎖プログラムは行番号 1000 から実行されます。

3
```
CHAIN "A:PROG1",1000,ALL
```

この例では，ファイル指定，行，および ALL を指定しています。変数型宣言などの影響は受けませんが，すべての変数が渡されます。連鎖プログラムは行番号 1000 から実行されます。

4
```
CHAIN MERGE "A:OVELAY",1000,DELETE 1000-5000
```

この例では，ALL 以外のすべての項目を指定しています。現プログラムの行番号 1000 ～ 5000 が削除された後，連鎖プログラムが組み込まれます。連鎖プログラム内には，変数型の宣言などは影響しますが，特に指定のない変数は渡りません。連鎖プログラムは行番号 1000 から実行されます。

CHDIR コマンド

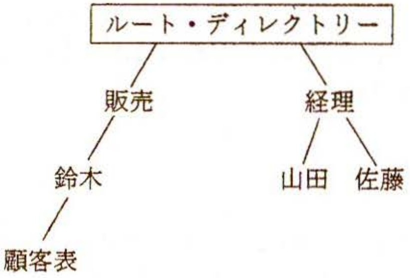
目的： ディスクの現行ディレクトリーを変更します。

構文： CHDIR — パス名 —■

説明： **パス名**には，63 文字以内のストリング式を指定します。現行ドライブ以外のドライブのディレクトリーを変更する場合，「ドライブ指定子：パス名」の形式で指定します。

このコマンドは，DOS の CHDIR コマンドと同じ効果をもちます。

例： 次の「木構造ディレクトリー」を例にとります。



CHDIR"販売¥鈴木¥顧客表"	ルート・ディレクトリーから顧客表に変える
CHDIR".."	顧客表から鈴木に変える

直上のディレクトリーは".."で指定します。

CHDIR"¥"	鈴木からルート・ディレクトリーに変える
CHDIR"経理"	ルート・ディレクトリーから経理に変える
CHDIR"山田" CHDIR"B:在庫"	現行ドライブ（A:）は山田，ドライブBは在庫を現行ディレクトリーとする

C

これにより，ドライブAを指定するとディレクトリー山田のファイルが参照され，ドライブBを指定するとディレクトリー在庫のファイルが参照されます。

CHR$ 関数

目的：　連続コードを，対応する文字に変換します。

構文：

```
         ┌…… , ……┐
―― CHR$ （ ─┴─ n ───┘─ ）――
```

説明：　nには，0～11535（連続コードの範囲）の整数式を指定します。カンマ（,）で区切って繰り返し指定することもできます。nを1つ指定すれば1文字，複数指定すれば対応する文字数のストリングを返します（連続コードについては，☞ 解説編　付録D「コード体系」）。

一般にこの関数は，画面やプリンターに制御文字を送る場合に利用されます。たとえば，内部スピーカーを鳴らすBEL文字（連続コード7）をPRINTステートメントに指定するためには，CHR$（7）と記述します。

文字から連続コードに変換するためには，ASC関数を使用します。

例：

1

```
Ok
PRINT CHR$(66),CHR$(2005)
B      楽
Ok
```

2

```
Ok
KEY1,"AUTO"+CHR$(13)
Ok
```

ストリング"AUTO"と復帰文字（連続コード13）を PF1 キーに定義しています。つまり，PF1 キーを押すと，AUTOを表示して改行するように設定しています。

3　次の例は，1バイト・コード文字と2バイトコード文字の，連続コードと文字の対応表を表示するプログラムです。

```
100 '--- 1バイト・コード文字 ---
110 SCREEN 0:CLS 0:KEY OFF
120 PRINT
130 PRINT "          ＊＊＊          1バイト・コード文字表示例 ＊＊＊"
140 PRINT
150 FOR I=32 TO 128:GOSUB 180:NEXT I
160 PRINT:PRINT
170 END
180 COLOR 0,0
190 PRINT USING "####";I;
200 COLOR 8,0:PRINT " ";CHR$(I);" ";
210 COLOR 0,0
220 IF POS(0)>74 THEN PRINT
230 RETURN
Ok
RUN

          ＊＊＊       1バイト・コード文字表示例       ＊＊＊

  32      33 !    34 "    35 #    36 $    37 %    38 &    39 '    40 (    41 )    42 *
  43 +    44 ,    45 -    46 .    47 /    48 0    49 1    50 2    51 3    52 4    53 5
  54 6    55 7    56 8    57 9    58 :    59 ;    60 <    61 =    62 >    63 ?    64 @
  65 A    66 B    67 C    68 D    69 E    70 F    71 G    72 H    73 I    74 J    75 K
  76 L    77 M    78 N    79 O    80 P    81 Q    82 R    83 S    84 T    85 U    86 V
  87 W    88 X    89 Y    90 Z    91 [    92 ¥    93 ]    94 ^    95 _    96 `    97 a
  98 b    99 c   100 d   101 e   102 f   103 g   104 h   105 i   106 j   107 k   108 l
 109 m   110 n   111 o   112 p   113 q   114 r   115 s   116 t   117 u   118 v   119 w
 120 x   121 y   122 z   123 {   124 |   125 }   126 ‾   127     128
```

```
10 '--- 2バイト・コード文字 ---
20 SCREEN 0:CLS 0:KEY OFF
30 PRINT
40 PRINT "          ＊＊＊        2バイト・コード文字表示例        ＊
＊＊"
50 PRINT
60 FOR I=&K0101 TO &K0194:GOSUB 110:NEXT I
70 PRINT:PRINT
80 FOR I=&K1601 TO &K1694:GOSUB 110:NEXT I
90 PRINT
100 END
110 COLOR 0,0
120 PRINT USING "####";I;
130 COLOR 8,0:PRINT " ";CHR$(I);"  ";
140 COLOR 0,0
150 IF POS(0)>72 THEN PRINT
160 RETURN
Ok
RUN
```

```
          ＊＊＊        2バイト・コード文字表示例        ＊＊＊

 256     257 、 258 。 259 ， 260 ． 261 ・ 262 ： 263 ； 264 ？ 265 ！
 266 ゛ 267 ゜ 268 ´ 269 ｀ 270 ¨ 271 ＾ 272 ￣ 273 ＿ 274 ヽ 275 ヾ
 276 ゝ 277 ゞ 278 〃 279 仝 280 々 281 〆 282 〇 283 ー 284 ― 285 ‐
 286 ／ 287 ＼ 288 ～ 289 ‖ 290 ｜ 291 … 292 ‥ 293 ‘ 294 ’ 295 “
 296 ” 297 （ 298 ） 299 〔 300 〕 301 ［ 302 ］ 303 ｛ 304 ｝ 305 〈
 306 〉 307 《 308 》 309 「 310 」 311 『 312 』 313 【 314 】 315 ＋
 316 － 317 ± 318 × 319 ÷ 320 ＝ 321 ≠ 322 ＜ 323 ＞ 324 ≦ 325 ≧
 326 ∞ 327 ∴ 328 ♂ 329 ♀ 330 ° 331 ′ 332 ″ 333 ℃ 334 ￥ 335 ＄
 336 ¢ 337 £ 338 ％ 339 ＃ 340 ＆ 341 ＊ 342 ＠ 343 § 344 ☆ 345 ★
 346 ○ 347 ● 348 ◎ 349 ◇

1666 亜 1667 唖 1668 娃 1669 阿 1670 哀 1671 愛 1672 挨 1673 姶 1674 逢 1675 葵
1676 茜 1677 穐 1678 悪 1679 握 1680 渥 1681 旭 1682 葦 1683 芦 1684 鯵 1685 梓
1686 圧 1687 斡 1688 扱 1689 宛 1690 姐 1691 虻 1692 飴 1693 絢 1694 綾 1695 鮎
1696 或 1697 粟 1698 袷 1699 安 1700 庵 1701 按 1702 暗 1703 案 1704 闇 1705 鞍
1706 杏 1707 以 1708 伊 1709 位 1710 依 1711 偉 1712 囲 1713 夷 1714 委 1715 威
1716 尉 1717 惟 1718 意 1719 慰 1720 易 1721 椅 1722 為 1723 畏 1724 異 1725 移
1726 維 1727 緯 1728 胃 1729 萎 1730 衣 1731 謂 1732 違 1733 遺 1734 医 1735 井
1736 亥 1737 域 1738 育 1739 郁 1740 磯 1741 一 1742 壱 1743 溢 1744 逸 1745 稲
1746 茨 1747 芋 1748 鰯 1749 允 1750 印 1751 咽 1752 員 1753 因 1754 姻 1755 引
1756 飲 1757 淫 1758 胤 1759 蔭
```

CINT 関数

目的：　数式を整数に変換します。

構文：　── CINT（x）──

説明：　x には，数式を指定します。この関数は，x の小数点以下を四捨五入した値を返します。

x の値が-32768 ～ 32767 の範囲を超えると，Overflow（桁あふれ）エラーとなります。

FIX 関数と INT 関数は，この関数と同様に整数を返します。また，CDBL 関数と CSNG 関数は，数値の精度を変換します。これらの関数の説明も，参考にしてください。

例：

```
Ok
PRINT CINT(45.67)
 46
Ok
PRINT CINT(-2.89)
-3
Ok
```

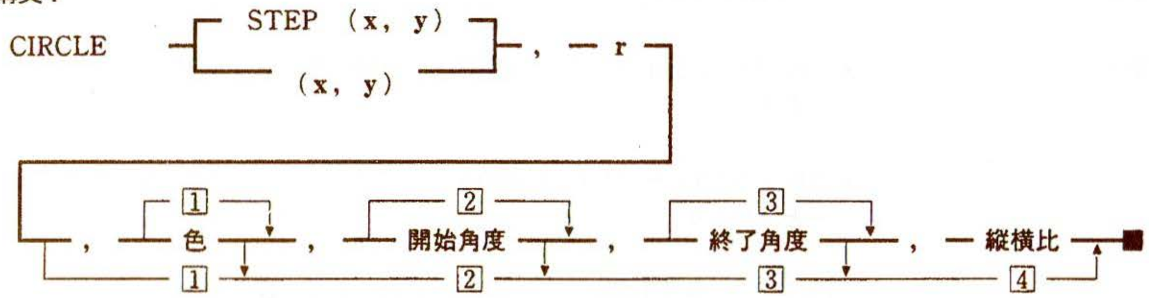
負数の四捨五入に注意してください。

CIRCLE ステートメント

目的：

画面に楕円（円）を描きます。グラフィック・モードでのみ使用できます。

構文：



[1] 前景色
[2] 0
[3] 2*3.141593
[4] 単色/16 色カラーは 1，4 色カラーは 2

説明：

（x，y）は，楕円の中心座標を絶対形式で表したもので，STEP（x，y）は中心座標を相対形式で表したものです。

半径は楕円の長軸半径を示し，画面上の点の数（ドット数）で指定します。

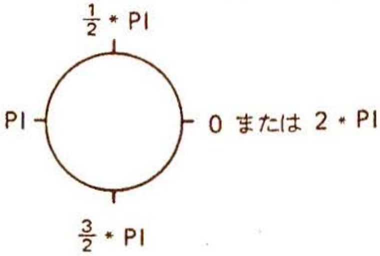
色には，楕円の色をパレット番号で指定します。

画面モード	パレット番号
単色	(0/1)
4 色カラー	0 ～ 3
16 色カラー	0 ～ 15

色を省略すると，前景色が使用されます。

開始角度と**終了角度**には，楕円の描き始めと終わりの位置を，角度（ラジアン）で指定します。指定できる値の範囲は，-2 π ～ 2 π です（π=3.141593 とします)。**開始角度**を省略すると 0，**終了角度**を省略すると 2 π が使用されます。

2 つの角度は，中心点から水平に右方向を基準として，反時計回りに計ります（つまり，右は 0 または 2 π，真上は π 2 となります)。角度に負の値を指定すると，その絶対値が使用されます。ただし，負の角度で指定された点と楕円の中心点の間には，直線が引かれます。**開始角度**>**終了角度**となる値を指定することも可能です。



縦横比は，楕円の縦横のドット数の比率を数式で指定します。**縦横比**が 1 より小さい場合，**半径**の値は x 方向のドット数となり，1 より大きい場合は y 方向のドット数となります。

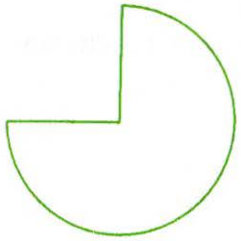
単色と 16 色カラーは，x 方向ドット間隔と y 方向ドット間隔が 1:1 に対応しています。ただし，4 色カラーのときは，x 方向ドット間隔が単色および 16 色カラー時の 2 倍になります。したがって，4 色カラー時の x 方向半径は，y 方向半径と同じドット数のときに 2 倍の距離として表示されます。

描いた楕円のうち，画面（ビューポート）からはみ出す部分は表示されません。また，CIRCLE 実行後の最終座標は，楕円の中心座標となります（VIEW ステートメント参照）。

円の大部分が画面からはみ出すような大きな円弧を描いた場合，計算誤差によって円弧が歪む場合があります。

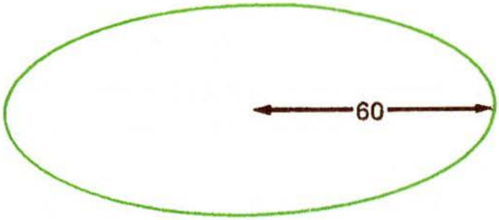
1

```
10 PI=3.141593
20 SCREEN 1,0  '--- 単色グラフィック・モード
30 CIRCLE (160,100),60,,-PI,-PI/2  '--- 円の一部を描く
```



2

```
10 SCREEN 1,0  '--- 単色グラフィック・モード
20 CIRCLE (160,100),60,,,,7/20  '--- 楕円を描く
```



3 次のプログラムは顔を描きます。

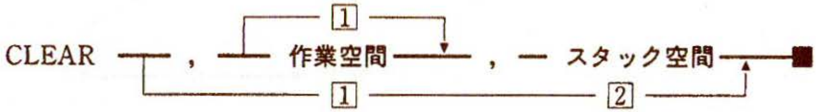
```
10 PI=3.141593
20 SCREEN 1,1
30 CIRCLE(140,150),10,15
40 CIRCLE(180,150),10,15
50 FOR HEAD=1 TO 100 STEP 10
60 CIRCLE(110+HEAD,100),30,6,,,3
70 NEXT:CIRCLE(160,157),30,3,1.4*PI,-1.6*PI
80 CIRCLE(160,160),50,2,1.4*PI,1.6*PI
```

CLEAR コマンド

目的：

すべての数値変数をゼロに，文字変数を空ストリングに設定します。

構文：



[1] 64K バイト
[2] 使用可能な記憶域の 1/8 または 512 バイトのいずれか小さい方

説明：

作業空間は，BASIC インタープリターの作業領域の最大限度を，バイト数で指定します。作業領域を小さく設定することは，作業領域外に使用空間を確保する場合に役立ちますが，本システムの場合充分な記憶域があるため，通常このパラメーターを使用する必要はありません。

スタック空間は，BASIC 用に確保するスタックの大きさを，バイト数で指定します。これは，プログラム内で GOSUB のネストや FOR..NEXT ループを多用したり，複雑な画面表示を行う場合に役立ちます。**スタック空間**を省略すると，使用可能な記憶域の 1/8 または 512 バイトのうち，いずれか小さい方のバイト数がスタックとして確保されます。

CLEAR は，データに使用されている記憶域を解放します（プログラム自身は消えません）。CLEAR の実行により，それまでのプログラムの各要素は，次の影響を受けます。

- 数値変数…………………………………… 0 に設定
- ストリング変数………………………… 空ストリングに設定
- 配列の定義（DIM）…………………… 無効
- 配列の基底（OPTION BASE）… 0 に設定

- 変数型定義，関数定義など…………… 無効

(DEFINT,DEFDBL,DEFSEG,DEFSTR,DEF FN,DEF SEG, DEF USR)

- 音楽（PLAY）…………………………前景に設定

その時点で鳴っていた音は止まる

記憶域の一部分だけを解放する場合，ERASE ステートメントを使用します。これは，指定した配列だけが，記憶域から消去されます。

例：

1 `CLEAR`

記憶域から，すべてのデータを消去します。

2 `CLEAR , 32768`

すべてのデータを消去した後，作業空間用に 32K バイトを確保します。

3 `CLEAR , , 2000`

すべてのデータを消去した後，スタック用に 2000 バイトを確保します。

4 `CLEAR, 32768, 2000`

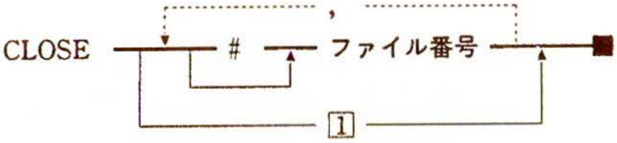
すべてのデータを消去した後，作業空間用に 32K バイトを，スタック用に 2000 バイトを確保します。

C

CLOSE ステートメント

目的： ファイル（または装置）の入出力を終了します。

構文：



[1] オープンしているすべてのファイルと装置をクローズ

説明： **ファイル番号**には，OPEN ステートメントで使用した番号を指定します。**カンマ（,）**で区切って繰り返すことにより，複数のファイルを指定することも可能です。

ファイル番号を省略すると，その時点でオープンしているファイルと装置はすべてクローズされます。

CLOSE ステートメントは，特定ファイル（または装置）とファイル番号との関係を切り離します。CLOSE 実行後は，再度 OPEN で何らかのファイル（または装置）に割り当てない限り，このファイル番号を使った入出力操作は無効となります。

順次出力のファイル（または装置）に CLOSE を実行すると，最終バッファーのデータが書き出されます。

END，NEW，RESET，SYSTEM，および R オプションなしの RUN は，オープンしているすべてのファイルと装置をクローズする働きをもちます。STOP は，この働きをもちません。

ファイルのオープンについては，OPEN ステートメントを参照してください。

例：

```
100 CLOSE 1,#2,#3
```

この例は，ファイル番号 1,2,3 のファイルと装置をクローズします。

```
200 CLOSE
```

この例は，ファイルと装置をすべてクローズします。

C

CLS ステートメント

目的： 画面を消去します。

構文：

[1] 下表を参照

説明： nは消去する領域を0～2の数値で指定します。nの指定による領域の消去は次のようになります。

ビューポートが有るとき	
CLS 0	画面全体
CLS 1	ビューポート内のすべて
CLS 2	文字表示範囲のみ
CLS	ビューポート内のすべて
[Clear]キー機能	文字表示範囲のみ

ビューポートが無いとき	
CLS 0	文字表示範囲のみ
CLS 1	文字表示範囲のみ
CLS 2	文字表示範囲のみ
CLS	画面全体
[Clear]キー機能	画面全体

表中のビューポートは VIEW ステートメントにより定義した画面中の図形表示領域です。この表中のビューポートは VIEW ステートメントで明示的に範囲を指定したものを指します。SCREEN ステートメントで画面モードを指定したときの画面や「VIEW」とだけ指定した場合は該当しません。

また表中の文字表示範囲は WIDTH ステートメントや VIEW PRINT ステートメントで指定した範囲です。

文字モードもグラフィック・モードの場合も画面は背景色になります（カラー・グラフィック・モードの場合，背景色はパレット番号 0 の色です）。

グラフィック・モードでは，最終座標が画面の中央に移動します。ビューポートを指定している場合，最終座標はビューポートの中央に移動します。

COLOR ステートメント

目的：

COLOR ステートメントは次の指定を行います。

- **単色文字モード/単色グラフィック・モード**

  単色文字モードのときは，文字の下線づけ，高輝度，明滅，反転，罫線の指定を行います。

  単色グラフィック・モードのときは，文字の下線づけ，反転，罫線の指定を行います。

- **カラー文字モード**

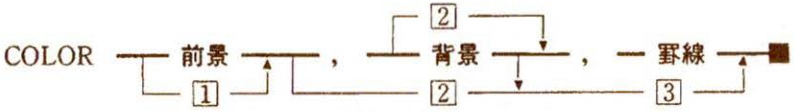
  文字の色（および反転），罫線の色，罫線の指定を行います。

- **カラー・グラフィック・モード**

  背景の色，標準パレット，罫線の指定，文字の色，罫線の色指定を行います。4 色カラー・グラフィック・モードと 16 色カラー・グラフィック・モードの色指定を行います。

COLOR ステートメント
単色文字/単色グラフィック・モード

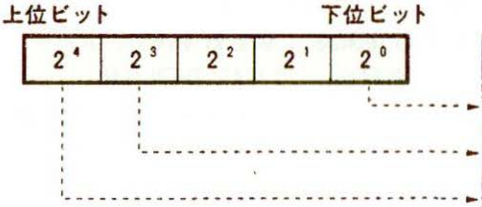
構文：



[1] [2] [3] 前の値

説明：

前景は文字の属性を 0 ～ 31 の数値で指定します。文字モードで指定できる文字の属性は下線づけ，高輝度，明滅です。グラフィック・モードで指定できる文字の属性は下線づけです。**前景**の値 0 ～ 31 は 5 桁の 2 進数で表現できます。この 5 桁のうち，2 の 0 乗，2 の 2 乗，2 の 4 乗の各桁が 0 か 1 によつて各属性の指定をします。各桁の属性との対応は下記のとおりです。



属性	重み
下線づけ	1
高輝度	8
明滅	16

対応する桁に 0 の値が入るとその指定はリセットされます。たとえば下線づけをしてその文字を明滅させるときには，下線づけと明滅に対応する桁に 1 をたてます。上記の表から重みを加えると 1 + 16=17 になります。**前景**にはこの 17 を指定します。

背景は文字の反転を 0 または 1 で指定します。**背景**が 1 のときに文字は反転になります。

罫線は次の意味を持つ値 0 ～ 4 を指定します。

0: 罫線を引かない
1: 縦罫線
2: 横罫線
3: 縦罫線と横罫線
4: 現在の画面の属性（前景,背景,罫線）は変えない

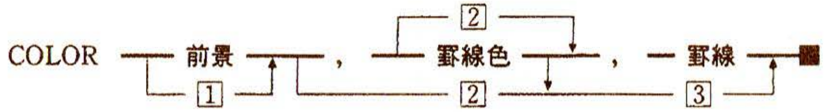
例:

```
10 SCREEN 0:COLOR 0,0:CLS
20 COLOR 0,0:PRINT "あなたの";
30 COLOR 8,0      '---------------- 高輝度文字
40 PRINT "暗証番号";
50 COLOR 0,0:PRINT "を入力してください"
60 COLOR 16,0     '---------------- 明滅文字
70 PRINT : PRINT "===>";
80 COLOR 25,1     '---------------- 見えない文字
90 INPUT PASSWORD$ : COLOR 0,0
100 IF PASSWORD$ = "1234" THEN 140
110 COLOR 24,0    '---------------- 高輝度, 明滅文字
120 PRINT : PRINT "暗証番号が違います";:BEEP
130 COLOR 0,0:PRINT:PRINT:GOTO 20
140 COLOR 0,1     '---------------- 反転文字
150 PRINT:PRINT "*** 暗証番号が認識されました ***";
160 COLOR 0,0 :PRINT:PRINT
```

この例は文字に高輝度や明滅の指定をした例です。暗証番号として1234を入力すると次のステートメントを実行します。

COLOR ステートメント
カラー文字モード

構文：



[1] [2] [3] 前の値

説明：

この COLOR ステートメントで色を指定するためには SCREEN ステートメントでカラー文字モードが設定されている必要があります。

前景は文字の色と反転を 0 ～ 31 の数値で指定します。0 ～ 15 は文字の色を表す色番号です。文字の反転は色番号に 16 を足して指定します。

カラー文字モードは色番号 0 ～ 7 までの色と 8 ～ 15 までの色を同時に表示できません。たとえば，青（1）の文字を表示後，前景に 8 ～ 15 または 24 ～ 31 を指定し実行すると青（1）の文字は薄い青（1）になります。

罫線色は罫線の色を 0 ～ 15 の色番号で指定します。

罫線は次の意味を持つ値 0 ～ 4 を指定します。

0：罫線を引かない
1：縦罫線
2：横罫線
3：縦罫線と横罫線
4：現在の画面の属性（前景,罫線色,罫線）は変えない

例 1:

```
10 SCREEN 0,1          '--- カラー文字モード
20 FOR I = 0 TO 3
30 FOR J = 0 TO 7
40 COLOR J+(I*8)
50 PRINT "カラー文字";
60 NEXT J
70 FOR K = 1 TO 2000 : NEXT K : NEXT I
80 COLOR 7,0,0
```

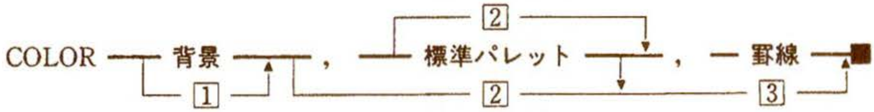
例 2:

次のプログラムは罫線属性を用いて表を作成した例です。

```
10 COLOR 7,0,0: SCREEN 0,1    '--- 属性の初期設定
20 LOCATE 1,23: PRINT"月別売上個数";
30 FOR I = 1 TO 5
40  COLOR ,3,3                      '--- 縦横の罫線を引く
50  FOR J=1 TO 4: LOCATE J+1,I*8+1: PRINT" ";: NEXT J
60  COLOR ,3,2                      '--- 横罫線を引く
70  FOR J=1 TO 4: LOCATE J+1,I*8+2: PRINT"       ";: NEXT J
80  LOCATE 6,I*8+1 : PRINT "        ";
90 NEXT I
100 COLOR ,3,1: FOR J=2 TO 5: LOCATE J,49: PRINT " ";: NEXT J
120 COLOR ,,4  '============= 文字を重ね書きする ==============
130 DATA 文房具,ノート,えんぴつ,けしごむ
140 DATA 12000,3500,3700,8500,2000,3300,11000,4700,4600,15000,5500,7800
150 FOR I=1 TO 4
160  LOCATE I+1,9: READ INDATA$: PRINT USING "&      &";INDATA$
170  LOCATE 2,11+8*I: PRINT I;
180 NEXT I
190 FOR I=1 TO 3
200  FOR J=1 TO 4
210   LOCATE 2+I,9+8*J: READ INDATA: PRINT USING "####,###";INDATA
220  NEXT J
230 NEXT I
```

COLOR ステートメント
カラー・グラフィック・モード（4色）

構文：



1 2 3 前の値

説明：

4色カラー・グラフィック・モードは16色の中の4色を同時に表示できるグラフィック・モードです。したがって，色番号は0～15，パレット番号は0～3が有効範囲です。**4色カラー・グラフィック・モードを使用するためには，**COLORステートメントやその他のグラフィック・ステートメントで4色カラーのモードを使用する前に**SCREENステートメントで4色カラー・グラフィック・モードを設定しておく必要があります。**各パレットに指定する色はPALETTEステートメントでも指定できます。

背景は背景の色を色番号0～15で指定します。**背景**の色番号はパレット番号0に設定されます。

標準パレットはパレット番号1～3に2つの標準パレットのいずれかを0または1で指定します（指定された値が偶数のときは0，奇数のときは1とみなされます）。

パレット番号	**標準パレット 0**	**標準パレット 1**
1	2（緑）	3（水色）
2	4（赤）	5（紫　）
3	6（黄）	7（白　）

文字と罫線の色はパレット番号3の色です。**背景**と**標準パレット**を指定するとパレット番号0～3に色番号が設定されます。これはPALETTEステートメントで各パレットに色を設定することと同じ意味になります。

罫線は次の意味を持つ値 0 〜 4 を指定します。

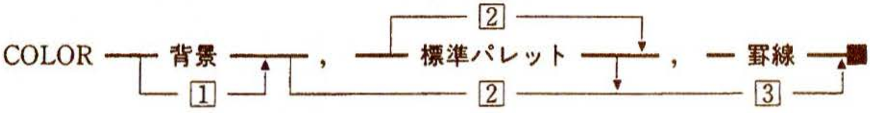
0: 罫線を引かない
1: 縦罫線
2: 横罫線
3: 縦罫線と横罫線
4: 現在の画面の属性（背景,標準パレット,罫線）は変えない

例：

```
10 '--- 4色カラー・グラフィック・モード ---
20 SCREEN 1,1
30 COLOR 0,1 : CLS
40 LINE -(RND*359,RND*511),INT(RND*4)
50 GOTO 40
```

COLOR ステートメント
カラー・グラフィック・モード（16 色）

構文：



[1] [2] [3] 前の値

説明：

16 色カラー・グラフィック・モードは 64 色の中の 16 色を同時に表示できるグラフィック・モードです。したがって，色番号は 0 ～ 63，パレット番号は 0 ～ 15 が有効範囲です。**16 色カラー・グラフィック・モードを使用するためには，**COLOR ステートメントやその他のグラフィック・ステートメントで 16 色カラーのモードを使用する前に **SCREEN ステートメントで 16 色カラー・グラフィック・モードを設定しておく必要があります。**各パレットに指定する色は PALETTE ステートメントでも指定できます。

背景は背景の色を色番号 0 ～ 15 で指定します。**背景**の色番号はパレット番号 0 に設定されます。

標準パレットはパレット番号 1 ～ 15 に指定する色を標準パレットの番号 0 または 1 で指定します（指定された値が偶数のときは 0，奇数のときは 1 とみなされます）。

パレット番号	標準パレット 0	標準パレット 1
1	36（明るい赤）	9（明るい青）
2	20（茶色）	18（明るい緑）
3	38（やまぶき色）	27（明るい水色）
4	54（明るい黄）	36（明るい赤）
5	34（黄緑）	45（明るい紫）
6	26（青緑）	54（明るい黄）
7	19（空色）	63（明るい白）
8	3（水色）	56（灰色）
9	11（緑青）	57（薄い青）
10	57（薄い青）	58（薄い緑）
11	41（すみれ色）	59（薄い水色）
12	45（明るい紫）	60（薄い赤）
13	37（濃いピンク）	61（薄い紫）
14	44（赤紫）	62（薄い黄）
15	63（明るい白）	63（明るい白）

罫線は次の意味を持つ値 0 ～ 4 を指定します。

0: 罫線を引かない
1: 縦罫線
2: 横罫線
3: 縦罫線と横罫線
4: 現在の画面の属性（背景,標準パレット,罫線）は変えない

文字と罫線の色はパレット番号 15 の色です。背景と標準パレットを指定するとパレット番号 0 ～ 15 に色番号が設定されます。これは PALETTE ステートメントで各パレットに色を設定することと同じ意味になります。

例：

次のプログラムは 16 色カラー・グラフィック・モードにおいて標準パレットの切り換えを行った例です。

```
10 KEY OFF : SCREEN 2,1 : LOCATE,,0 : ID = 1
20 '--- パレットの絵を表示
30 LINE(52,58)-(702,348),15,B : C = 0
40 FOR Y = 116 TO 203 STEP 87
50   FOR X = 104 TO 650 STEP 78
60     CIRCLE(X,Y),29
70     PAINT(X,Y),C,15 : C=C+1
80 NEXT X,Y
90 CIRCLE(611,290),60,,,,3/5
100 LOCATE 1,1 : PRINT "※ 標準パレット 1"
110 LOCATE 24,1
120 PRINT "どれかキーを押してください。";
130 PRINT "パレットの色が入れ替わります。";
140 '--- キーを押すごとに標準パレットを入れ替える
150 POKE 106,0
160 K$ = INKEY$ : IF K$ = "" THEN 160
170 ID = ID + 1 : P=ID MOD 2
180 IF P = 0 THEN LOCATE 1,1 : PRINT "※ 標準パレット 0"
190 IF P = 1 THEN LOCATE 1,1 : PRINT "※ 標準パレット 1"
200 COLOR 0,P
210 GOTO 150
```

COM（n）ステートメント

目的：

通信による割り込み処理を，可能な状態，無効な状態，または停止状態に設定します。

構文：

COM（n）——ON——■
COM（n）——OFF——■
COM（n）——STOP——■

説明：

nには，通信カードの番号（1または2）を指定します。

COM（n） ONは，通信による割り込み処理を可能にします。

ON COM（n）を有効にするためには，データや信号などの文字が通信カードに送られる前に，COM（n） ONを実行する必要があります。

0以外の行番号をもつON COM（n）がある場合，COM（n） ON実行後は，ステートメントを1つ実行するたびに，通信カードに文字が入っているか否かの検査を行います。

COM（n） OFFは，通信による割り込み処理を無効にします。

COM（n） OFFの実行後は，通信カードに文字が入っても割り込み処理は実行されません。また，通信カードの活動情報（つまり何らかの文字が送られていること）も記憶されません。

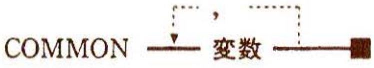
COM（n） STOPは，通信による割り込み処理を停止します。

COM（n） STOPの実行後は，通信カードに文字が入っても割り込み処理は実行されません。ただし，通信カードの活動情報（文字が送られていること）が記憶されるため，COM（n） ONを実行した時点で直ちに割り込み処理が行われます。

COMMON ステートメント

目的： 連鎖されたプログラムに変数を渡します。

構文：



説明： **変数**は，CHAIN で連鎖されたプログラムに渡す変数の名前です。カンマ(,)で区切って任意の数だけ指定することができます。配列変数は，変数名の後に（）を付け加えて指定します。

COMMON ステートメントは，プログラム内のどこに置いても同じ効果をもちます（ただし，プログラムを分かりやすくするために，最初に置くことをお勧めします）。

複数の COMMON ステートメントを同一プログラム内で使用することはできますが，同じ変数を複数の COMMON ステートメントに指定することはできません。

COMMON で渡された配列変数については，連鎖プログラム内で配列の次元を宣言する必要はありません。

すべての変数を連鎖プログラムに渡す場合は，COMMON ではなく，ALL 付きの CHAIN ステートメントを使用してください。

例：

```
100 COMMON A,BEE1,C,D(),G$
110 CHAIN "APROG3"
```

この例では，ドライブ A の PROG3 プログラムを連鎖して，変数 A,BEE1 ,C，および G$ と，配列変数 D を渡します。

CONT コマンド

目的： プログラムの実行を再開します。

構文： CONT ——■

説明： このコマンドは，次のいずれかの原因で中断されたプログラムの実行を，再開させます。

- Break キー機能
- STOP ステートメント，または END ステートメント

プログラムは，中断の起きた次のステートメントから再開されます。ただし，INPUT ステートメントによるプロンプト表示の直後に中断が起こった場合，プログラムは再度プロンプトを表示した後に再開されます。

直接モードのコマンドで変数の値を表示，修正しても，プログラム自身は修正されません。

中断中にプログラムの変更を行うと，CONT コマンドによる実行の再開はできません。

これを応用すれば，プログラムの任意の箇所の STOP ステートメントで中断させることでデバックが効果的に行えます。

例：

次の例は，1 から 50 までの数字を表示するプログラムです。この表示を Break キー機能で中断させ，CONT コマンドで再開させています。再開は，中断の起きた次の行から行われています。

```
10 FOR A=1 TO 50
20 PRINT USING "###";A;
25 IF POS(0)>30 THEN PRINT
30 NEXT A
Ok
RUN
  1  2  3  4  5  6  7  8  9 10
 11 12 13 14 15 16 17 18 19 20
 21 22 23 24 25 26
Break in 20
Ok
CONT
 27 28 29 30 31 32 33 34 35 36
 37 38 39 40 41 42 43 44 45 46
 47 48 49 50
Ok
```

C

COS 関数

目的：　角度の余弦を返します。

構文：　── COS (x) ──

説明：　x には，ラジアンで表した角度を，数式で指定します。

BASIC は π を 3.141593 として計算するため，度からラジアンへの数値の変換には，プログラムで 3.141593/180 を掛けてください。

この関数は，単精度で計算した値を返します。BASIC を始動するときに/D を指定すると倍精度で計算した値を返します。

例：　次のプログラムは，π の余弦 (-1) を表示した後，180 度の余弦 (-1) を表示します。度はラジアンに変換した上で COS 関数に指定しています。

```
10 PI=3.141593
20 PRINT COS(PI)
30 DEGREES=180
40 RADIANS=DEGREES*PI/180
50 PRINT COS(RADIANS)
Ok
RUN
-1
-1
Ok
```

CSNG 関数

目的：　数式を単精度の数値に変換します。

構文：　—— CSNG (x) ——

説明：　xには，数式を指定します。

数値の精度変換の規則については，☞解説編　第2章の「数値精度の変換」。

数値を精数に変換するCINT関数，および倍精度に変換するCDBL関数の記述も参照してください。

例：

```
10 A#=975.3421222#
20 PRINT A#;CSNG(A#)
Ok
RUN
 975.3421222  975.3421
Ok
```

この例は倍精度のA#と，四捨五入で8桁（単精度）にしたものを表示しています。

C

CSNG$ 関数

目的：

ストリング内の2バイト・コードの英数字を，すべて1バイト・コードに変換します。

構文：

—— CSNG$ (**x$**) ——

説明：

x$には，ストリング式を指定します。

変換されるものは，2バイト・コード（連続コード256以上）の数字，英小文字，および英大文字であり，それぞれ1バイト・コード（連続コード255以下）の数字，英小文字，および英大文字になります。これ以外の文字は，変換されません。

1バイト・コードの英数字を2バイト・コードに変換するためには，CDBL$ 関数を使用します。

例：

```
10 PRINT CSNG$("０１２３４５６７８９")
20 PRINT CSNG$("－３．１４１５９２６５４")
Ok
RUN
0123456789
-3.141592654
Ok
```

CSRLIN 変数

目的： 画面上の，カーソルの行位置を返します。

構文： —— CSRLIN ——

説明： この変数は，カーソルの現在の行位置を示す値を返します。返される値の範囲は WIDTH で指定した行数に依存します。

カーソルの現在の桁位置を求めるためには，POS 関数を使用します。

カーソルを特定の行に設定するためには，LOCATE ステートメントを使用します。

例： 次のプログラムは，現在のカーソルの位置を X と Y に格納し，カーソルを移動して別の位置に文字を表示した後，再び元の位置にカーソルを戻しています。

```
10 CLS
20 Y=CSRLIN      '----- カーソルの行位置を求める
30 X=POS(0)      '----- カーソルの桁位置を求める
40 '--- 23行目, 1桁目に移動する ---
50 LOCATE 5,1:PRINT "カーソル位置";
60 LOCATE Y,X    '----- カーソル位置を元に戻す
Ok
```

C

CVI、CVS、CVD 関数

目的：　数字を，ストリング型から数値型に変換します。

構文：

— CVI （2バイト・ストリング） —
— CVS （4バイト・ストリング） —
— CVD （8バイト・ストリング） —

説明：　上記 3 つの関数は，それぞれ次の長さのストリング型の数字を変換します。

関数	ストリング長さ	変換後の型
CVI	2バイト	整数
CVS	4バイト	単精度
CVD	8バイト	倍精度

これら 3 つの関数は，実際のデータの型を変換するのではなく，BASIC がデータを扱う際の解釈をストリング型から数値型に変えることを指定するものです。

通常，CVI，CVS，CVD 関数は，MKI$，MKS$，MKD$ 関数と組みで使用します。MKI$，MKS$，MKD$ は，数値を内部形式のままストリングの領域へ複写します。それに対して CVI，CVS，CVD は，それらのストリングを数値変数の領域に複写します。

ランダム・ファイルはデータを必ずストリング型で記憶しますが，これらの関数を用いると数値を内部形式のまま保管できるため，ファイルの数値データ領域を節約できます。

ストリング型の数字の内部形式を数値に変換するときは，VAL 関数を使用してください。

例：

```
50 FIELD #1,4 AS N$,12 AS B$
60 GET #1
70 Y=CVS(N$)
```

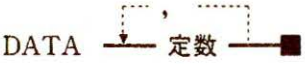
この例では，行 70 でランダム・ファイル（#1）のフィールドを定義し，行 80 でレコードを読み込みます。行 90 で，最初の 4 バイトのデータを単精度の数値に変換しています。

C

DATA ステートメント

目的：　READ ステートメントで読み込む定数を指定します。

構文：



説明：　**定数**には，数値定数，またはストリング定数を指定します（式は指定できません）。**定数**は，カンマ（,）で区切って任意の数だけ繰り返し指定することができます。

数値定数は，整数，固定小数点数，浮動小数点数，16 進数，18 進数，JIS16 進定数，または JIS 区点定数のいずれの形式も使用できます。

ストリング定数は，カンマ（,），コロン（:），先行ブランク，または後続ブランクを含む場合，二重引用符（" "）で囲って指定します。

このステートメントは，プログラム内のどこにでも，またいくつでも指定することができます。複数の DATA ステートメントがある場合，行番号の順番に続けてアクセスされます。

READ ステートメント中の変換型の指定（数値またはストリング）が，DATA ステートメント中の対応する定数と一致しない場合，Syntax error（構文エラー）が起こります。

一度使用した DATA ステートメントの定数を再利用するためには，RESTORE ステートメントを指定します。

DATA ステートメントには，アポストロフィーで注釈をつけることができません。

例：　「READ ステートメント」の例を参照してください。

DATE$ 変数およびステートメント

目的： 日付の設定，または読み取りを行います。

構文：

—— DATE$ ——

DATE$＝ —— x$ ——■

D

説明：

1 **変数**

この変数は，mm-dd-yyyy の形式で表された日付を返します。

mm は月を表す 2 桁の数字，dd は日を表す 2 桁の数字，yyyy は年を表す 4 桁の数字であり，それぞれを連結するハイフン（-）と共に 10 文字のストリングとして返されます。

2 **ステートメント**

x$ には，次のいずれかの形式で，日付を示すストリング式を指定します。

mm-dd-yy
mm/dd/yy
mm-dd-yyyy
mm/dd/yyyy

mm は，月を示す数字を 1 桁または 2 桁で指定します。1 桁だけ指定した場合，その数の前に 0 があるものと仮定されます。

dd は，日を示す数字を 1 桁または 2 桁で指定します。1 桁だけ指定した場合，その数の前に 0 があるものと仮定されます。

yy または yyyy は，年を示す数字を 1 桁，2 桁，または 4 桁で指定します。1 桁だけ指定した場合は指定した数の前に 200 が，2 桁指定した場合は指定した数の前に 00 ～ 77 までは 20 が，80 ～ 99 までは 19 があるものと仮定されます。3 桁を指定すると Illegal function call（違法関数呼出し）となります。4 桁の場合 1980 ～ 2079 まで指定できます。

例：

```
10 DATE$="8/4/86"
20 PRINT DATE$
Ok
RUN
08-04-1986
Ok
```

この例では，DATE$ 変数で日付に 1986 年 8 月 4 日を設定し，次に DATE$ ステートメントで日付を読み取っています。1 桁で設定した月が 2 桁で表示されていること，年に 19 がつけたされていること，および区切り文字がスラッシュからハイフンに変わっていることに注意してください。

DEF FN ステートメント

目的：

ユーザーが独自に作成した関数を定義します。

構文：

[1] 引数は無し

説明：

DEF と FN の間には必ずスペースを入れてください。

名前には，変数名を指定します。この**名前**は，前に FN が加わった形で関数名となります。

引数は，関数の定義に使用する変数です。プログラム中でこの関数を使用すると，各**引数**と対応するパラメーターを置き換えた後，**式**の値を求めます。引数は，最大 12 個まで指定できます。

式は，関数の返す値を定義します。**式**には，**引数**だけでなくプログラム中の変数も使用することができます。

引数に指定された変数名は，**式**の中だけで有効です。プログラムで同じ名前の変数を使用していても，それぞれ独自のものとして働きます。また，**引数**以外に**式**に使用した変数は，プログラム中の同じ名前をもつ変数が，そのまま用いられます。

名前で指定した変数の型は，この関数の型となります（解説編　第 2 章の「変数型」参照）。この型と**式**の型は，数値またはストリングのいずれかに一致させる必要があり，一致しない場合は Type mismatch（型の不一致）エラーになります。また，数値型の場合，式の値は関数の精度に変換された後，関数の呼び出し元に返されます。

DEF FN ステートメント実行前に関数を使用すると，Undefined user function（未定義関数）エラーが起こります。また，同じ関数名に複数の定義付けを行うと，最新の定義が有効となります。このステートメントは，直接モードでは無効です。

例：

1

```
10 PI=3.141593
20 DEF FNAREA(R)=PI*R^2
30 INPUT "半径は";RADIUS
40 PRINT "円の面積は";FNAREA(RADIUS)
Ok
RUN
半径は? 2
円の面積は 12.56637
Ok
```

このプログラムは，行 20 で円の面積を求める公式を関数に定義した後，行 40 でこれを使って計算しています。

2

```
10 DEF FNMD(X,Y)=X-(INT(X/Y)*Y)
20 A=FNMD(7.4,4)
30 PRINT A
Ok
RUN
 3.4
Ok
```

このプログラムでは，2 つの引数を使用しています。

DEF SEG ステートメント

目的：

記憶域の現在のセグメントを指定します。

構文：

```
DEF SEG ─┬─ =アドレス ─┬─■
         └──── [1] ────┘
```

[1] BASIC のデータ・セグメントの先頭アドレス

説明：

DEF と SEG の間には必ずスペースを入れてください。

アドレスは，0 ～ 65535 の範囲の数式で，16 バイト境界に基いた値を指定します。

アドレスを省略すると，BASIC のデータ・セグメント（DS），つまり作業空間の最初の部分が現在のセグメントとなります。BASIC 開始時に初期設定されるセグメントも同様に DS です。

範囲外の値を**アドレス**に指定すると，Overflow（桁あふれ）エラーが起こり，現在のセグメントは変更されません。

BLOAD，BSAVE，CALL，PEEK，POKE，DEF USR で指定するオフセットは，すべて現在のセグメント内での位置を表します。**アドレス**で指定した値が左に 4 ビットだけシフトされ（つまり 16 倍されて），これにオフセットで指定した値が加えられたものが実際のアドレスとなります。

DEF SEG で設定されたセグメントは，RUN コマンドで DS に再設定されることはありません。

DEF SEG に関する詳細な情報については☞ 解説編 付録 B「機械語サブルーチン」を参照してください。

DEF 型ステートメント

目的：

変数の型を示す文字を，整数，単精度，倍精度，またはストリングについて宣言します。

構文：

説明：

型には，INT，SNG，DBL，または STR を指定します（それぞれ整数，単精度，倍精度，またはストリングを示しています）。DEF と型は，間にスペースを入れずに続けて記述します。

文字は，1 字の英字（A ～ Z）を指定します。**文字-文字**は，最初の**文字**から 2 番目の**文字**までのすべての英字を意味します。

このステートメントを実行すると，**文字**，または**文字-文字**の英字から始まる名前をもつ変数は，**型**で指定された変数型をもちます。ただし，型宣言文字（%，!，#，または $）がつけたされた場合は，型宣言文字による型が優先されます（☞解説編　第 2 章「変数型」）。

DEF 型ステートメントに指定されていない文字を名前の最初にもつ変数は，すべて単精度型になります。

DEF 型ステートメントは，この型宣言が影響する変数を使用する前に，実行する必要があります。変数の使用後に DEF 型ステートメントで別な型を宣言すると，宣言した型が数値型なら 0，ストリング型なら空白文字に，変数の値が初期設定し直されます（それまでの変数の値は，すべて無効になります）。このような理由から，一般に DEF 型ステートメントはプログラムの最初に置きます。

例：

```
10 DEFDBL L-P
20 DEFSTR A
30 DEFINT X,D-H
40 ORDER=1#/3:PRINT ORDER
50 ANIMAL="CAT":PRINT ANIMAL
60 X=10/3:PRINT X
Ok
RUN
 .3333333333333333
CAT
 3
Ok
```

行 10 により ORDER は倍精度，行 20 により ANIMAL はストリング，行 30 により x は単精度と型宣言されています。

D

DEF USR ステートメント

目的：

機械語サブルーチンの開始アドレスを指定します。

構文：

DEF USR ─┬─ n ─┬─ =**オフセット** ──■
　　　　　└─ [1] ─┘

[1] 0

説明：

n は，複数の機械語サブルーチンを識別するための番号であり，0 ～ 9 の数字で指定します。USR 関数により機械語サブルーチンを呼び出すときも，この番号で識別します。n を省略すると，USR0 とみなされます。

オフセットには，0 ～ 65535 の範囲の整数式を指定します。この値は，現在のセグメントにおける開始位置を示します（☞ DEF SEG ステートメント）。

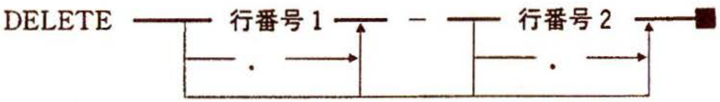
同じ n の値をもつ DEF USR ステートメントを，プログラム内にいくつも指定することができます。USR 関数は，常にその USRn に対する最新の DEF USR を使用して，機械語サブルーチンを呼び出します。したがって，使用後の USRn に別なサブルーチンのオフセットを指定することにより，いくつもの機械語サブルーチンを呼び出すことが可能です。詳細については，☞ 解説編　付録 B「機械語サブルーチン」を参照してください。

DEF USR ステートメントで指定していない USRn を USR 関数で呼び出すと，Illegal function call（違法関数呼出し）エラーとなります。

DELETE コマンド

目的： 指定した範囲の行を，プログラムから削除します。

構文：



説明： **行番号 1** は，削除する部分の最初の行番号を指定します。

行番号 2 は，削除する部分の最後の行番号を指定します。

ピリオド（.） は，現在行（最後に入力した行）の行番号を示します。「.－.」は現在行 1 行だけを意味します。

行番号 1 または**ピリオド**を省略すると，プログラムの最初の行番号が使用されます。

行番号 2 または**ピリオド**を省略すると，プログラムの最後の行番号が使用されます。また，**行番号 2** または**ピリオド**を指定する場合は，その前に必ずハイフン（－）を置きます。

指定された行番号は，削除の範囲に含まれます。また，プログラム内にない行番号を指定すると，Illegal function call（違法関数呼出し）エラーが起こります。

例：

DELETE 40-100	行 40 から行 100 までを削除します。
DELETE -.	最初から現在行までを削除します。
DELETE 40-	行 40 から最後までを削除します。

DIM ステートメント

目的：　配列変数の添字の最大値を指定し，記憶域を割り振ります。

構文：　DIM 変数（添字）［, 変数（添字）…］

説明：　**変数**には，配列に使用する変数名を指定します。

添字には，配列の添字の最大値を数式で指定します。2 次元以上の配列には，各次元の添字の最大値を，かっこ内にカンマ（,）で区切って指定します。

DIM ステートメントで指定していない変数名を，配列として添字付きで使用すると，この配列変数の添字の最大値は 10 とみなされます。また，添字の最小値は，OPTION BASE ステートメントで 1 を指定しない限り，0 となります（OPTION BASE ステートメント参照）。範囲外の添字を使用すると，Subscript out of range（範囲外の添字）エラーが起こります。

配列の次元は最大 255，各次元ごとの要素数の最大は次のとおりです。ただし，これらの値は記憶域の大きさとステートメントの長さにより制限を受けます。

整数型配列とストリング型配列… 32767
単精度実数型配列…………………… 16383
倍精度実数型配列…………………… 8191

同じ変数名に対して複数の DIM ステートメントを指定すると，Duplicate Definition（二重定義）エラーが起こります。ただし，ERASE ステートメントで消去された配列については，この限りではありません。

DIM ステートメントで指定された配列の各要素は，数値配列については 0，ストリング配列については空ストリング（可変長で長さ 0）が初期設定されます。

例：

次のプログラムは，SIS (0) ～ SIS (12) の 13 個の要素をもつ 1 次元の数値配列と，WRR$ (0,0) ～ WRR$ (2,2) の 9 個の要素をもつ 2 次元のストリング配列を作成しています。

```
10 WRRMAX=2
20 DIM SIS(12),WRR$(WRRMAX,2)
30 DATA 26.5,37,8,29,80,9.9,&H800
40 DATA 7,18,55,12,5,43
50 FOR I=0 TO 12
60 READ SIS(I)
70 NEXT I
80 DATA 紫式部,吉田兼好,"A:"
90 DATA "はい,そうです",こんにちわ,おはよう
100 DATA 東京都 港区,大阪,北海道
110 FOR I=0 TO 2
120 FOR J=0 TO 2
130 READ WRR$(I,J)
140 NEXT J,I
150 PRINT SIS(3);WRR$(2,0)
Ok
RUN
 29 東京都 港区
Ok
```

D

DRAW ステートメント

目的：

グラフィック・モードの画面に，直線で構成された図形を描きます。

構文：

DRAW——ストリング——■

説明：

ストリングには移動命令を表す特定の文字から構成されたストリング式を指定します。ストリング内の命令文字の機能を組み合わせて，1つの図形が描かれます（使用できる命令文字とその機能については，後述します）。ストリングは引用符で囲みます。

ストリングに指定する移動命令は，絵筆の移動を想定すると分かりやすくなります。各命令文字により，絵筆の移動する方向と距離（または移動先の位置），移動時に線を引くか否か，絵具の色などが指定できます。絵筆を置く位置を指定しなければ，それまでに描かれた図形の最後の絵筆の位置から描き始めます。

絵筆が最後に置かれた位置，言い換えると現在の絵筆の位置を最終座標と呼びます。最終座標は1つのDRAWステートメント内でも，命令文字による移動が起こるたびに変わります。

次に命令文字について説明します。命令文字は次のように分類することができます。

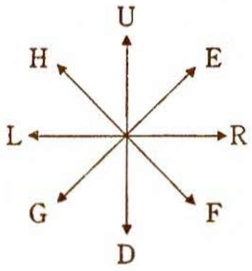
ストリングに指定する移動の命令

方向と距離の指定	Un,Dn,Ln,Rn,En,Fn,Gn,Hn
移動先座標の指定	Mx,y
移動の種類の指定	B,N
回転の命令	An,TAn
色の指定	Cn,P
倍率の指定	Sn
変数の指定	X

画面上の線は，移動の命令により描かれます。これには，方向と距離による指定と，移動先の位置による指定の 2 種類の指定の方法があります。いずれの方法でも，移動先が画面の範囲外となる命令は無効です。

- 方向と距離による移動の命令は，8 種類の英字と数値 n の組み合わせで指定します。距離は，画面の x 座標（水平方向）n 目盛分として表されます。

Un　上へ n 移動
Dn　下へ n 移動
Ln　左へ n 移動
Rn　右へ n 移動
En　右上方へ n 移動
Fn　右下方へ n 移動
Gn　左下方へ n 移動
Hn　左上方へ n 移動



- 移動の命令は移動先の絶対座標，または相対座標で指定します。

Mx,y　x の前に符号（＋または－）を付けると（x,y）は現座標からの相対座標を示します。x の前に符号が無ければ，（x,y）は絶対座標を示します。

通常は移動の命令を与えると絵筆は線を引きながら移動し線の終点でとまります。ただし，次に挙げる文字を移動の命令文字の前に付けると，線を引かずに絵筆を移動させたり，線を引いた後で元の位置に絵筆を戻しておくことができます。

B　線を引かずに移動することを指定します。

N　移動の後でもとの位置に戻る，つまり最終座標を元のままにしておくことを指定します。

方向と距離による移動の命令を使うと 8 方向への線引きが可能です。ただし，この命令の前に回転の命令を付加すると，自由な方向に線を引くことができます。移動の命令で与えられる方向に，回転角度を加えたものが実際の移動方向になります。

An　n は 90 度単位の回転角度を整数 0 〜 3 で指定します。方向は反時計回りです。

n	0	1	2	3
角度	0	90	180	270

TAn　n は反時計回りの回転角度を示す値で，-360 〜＋ 360 の範囲の整数です。n の値が正（＋）であれば反時計回りに，負（-）であれば時計回りに方向が変わります。

線引きの色を決めたり，描いた図形に色を塗るためには，次の命令文字を指定します。

Cn　線の色を決めます。n はパレット番号を指定します。C を指定しなければ，現在の前景色で線が引かれます。

画面モード	パレット番号
単色	(0/1)
4 色カラー	0 〜 3
16 色カラー	0 〜 15

Pn1,n2　n1 は図形を塗る色を，n2 は図形のふちの色をパレット番号で指定します。n1 を省略すると背景になります。n2 を省略すると Illegal function call（違法関数呼出し）エラーになります。

図形を拡大，または縮小することも可能です。図形の倍率，つまり移動距離の倍率は，次の命令文字で指定します。指定した倍率は，別な倍率を指定するまでの間，すべての移動に対して働きます。

Sn nは倍率を決める要素（スケール因子と呼ばれる）で1～255の整数で指定します。実際の倍率は，nを4で割った値となります。Sで指定しなければ1倍（n=4）です。

ストリングに指定する移動命令の一部分，また命令文字に付け加える数値は，変数としてDRAWステートメントの外部で値を設定することもできます。

X変数名; **ストリング**に指定する移動命令の一部を**変数名**のストリング型変数で与えることを意味します。したがって，DRAWの実行前に命令文字で構成されたストリングを変数に与えておきます（空ストリングも可能です）。また，一部を変数にすることにより，全体で256文字以上の**ストリング**を指定することもできます。

=変数名; 命令文字に付け加える数値（nなどを使って示したもの）を**変数名**の数値型変数で与えることを指定します。DRAWの実行前に，有効な数値を設定しておく必要があります。

また，**ストリング**の外で変数名を指定する方法もあります。ただし，この方法は，変数が**ストリング**の最後にある場合のみ使用できます。

"… X"+ VARPTR$（変数名）
「X変数名;」と同じ機能をもちます。

"… ="+ VARPTR$（変数名）
「=変数名;」と同じ機能をもちます。

D

例：

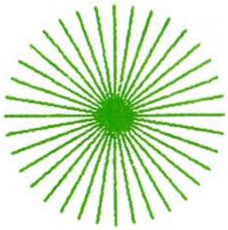
次のプログラムはどちらも同じ正方形を描きます。

```
10 SCREEN 2,0
20 A=100
30 DRAW "U=A;L=A;D=A;R=A;"
```

```
10 SCREEN 2,0
20 FOR X=0 TO 3
30 DRAW "A=X;U 100"
40 NEXT X
```

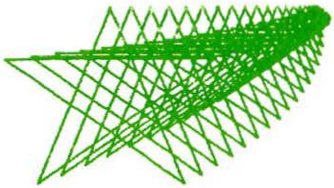
次のプログラムは放射線上の線を描きます。

```
10 SCREEN 1,0
20 DRAW "BM360,200"
30 FOR D=0 TO 360 STEP 10
40 DRAW "TA=D;NU200"
50 NEXT D
```



次のプログラムは流れ星を描きます。

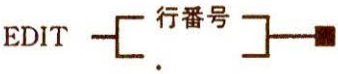
```
10 SCREEN 1,1:COLOR 0,0:CLS
20 DRAW "BM300,300"
30 STAR$="M+7,17 M-17,-12 M+20,0 M-17,12 M+7,-17"
40 FOR SCALE=1 TO 40 STEP 2
50 DRAW "C1;S=SCALE;BM-2,0;XSTAR$;"
60 NEXT
```



EDIT コマンド

目的： プログラム中の指定した行を表示します。

構文：



説明： 行番号には表示させる行の行番号を指定します。この行番号がプログラム中に存在しない場合，Undefined line number（行番号不定）エラーが起こります。

ピリオド（.）は直前に修正，または入力した行の行番号を示します。

このコマンドは特定のプログラム行を修正する場合に役に立ちます。

END ステートメント

目的：

プログラムの実行を終了し，すべてのファイルをクローズした後，直接モードの入力レベルに戻ります。

構文：

END ──■

説明：

END ステートメントは，プログラム内のどこにでも置くことができます。ただし，プログラムの最後では END の有無にかかわりなく，これらの終了処理が行なわれます。

STOP ステートメント（プログラムの実行を止める）と END ステートメントは，次の点で異なります。

- 実行が停止した時点で STOP は中断のメッセージを表示するが，END は表示しない。
- END はすべてのファイルをクローズするが，STOP はクローズしない。

例：

```
70 IF K>1000 THEN END ELSE GOTO 20
```

この例では，K の値が 1000 より大きい場合にプログラムが終了し，それ以外の場合は行 20 に分岐します。

ENVIRON ステートメント

目的：

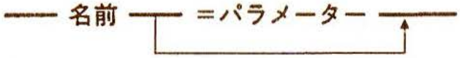
DOS のコマンド処理プログラムの環境の修正をします。

構文：

ENVIRON — **パラメーター指定** —■

説明：

パラメーター指定は，次の形のストリング式で指定します。



名前には，DOS のコマンド名（PATH や PROMPT），または BASIC 以外のプログラム（呼び出す）で使用するパラメーター名を，ストリング式で指定します。**パラメーター**には，コマンド名やパラメーター名に対する値を指定します。

名前と**パラメーター**は，等号（=）またはスペースで区切ります。2 つ目以降の等号やスペースは，すべて**パラメーター**の値の一部とみなされます。

値の設定を環境から削除する場合は，パラメーターに空ストリング（何も指定しない）かセミコロン（;）1 つを指定します。これにより，**名前**に関する情報が環境から削除されます。

このステートメントは DOS の SET コマンドと同様の働きをします（ただし，環境の表示機能はありません）。

環境に関する技術的な情報は，DOS の内部構造とプログラム・インターフェースを参照してください。

このステートメントは，プログラムから別のプログラムを呼び出す場合に，次のように利用することができます。

- 別プログラムの実行に移る前に，パスの省略時の値を換えておく（通常はDOSのPATHコマンドで変更する)。
- 呼び出されプログラム内で使用する独自のパラメーターの値を，もとプログラムから環境を介して受け渡す。

パラメーター指定がストリング式でない場合はType mismatch（型の不一致）エラーが起こり，環境の領域が一杯になった場合はOut of memory（記憶域不足）エラーが起こります。

例：

次の例はパスの省略時の値，つまりディレクトリー探索の設定を，「A:販売」「A:経理」の順に変えています。

```
ENVIRON "PATH=A:¥販売;A:経理"
```

次の例は別のプログラムで使用する引数P1に対して，呼び出し時に受け渡す値を設定しています。

```
70 ENVIRON "P1=0"
```

次の例はすでに環境に設定されている"P1"という項目の情報を削除します。

```
ENVIRON "P1=;"
```

ENVIRON$ 関数

目的： DOS のコマンド処理プログラムの環境から，特定の項目に設定された値を読み取ります。

構文：

— ENVIRON$ —（ 名前 | n ）—

説明： **名前**には，呼び出す項目名（DOS コマンド名またはパラメーター名）をストリング式で指定します。

名前には環境に記述されている項目名のとおりに指定してください。環境では，大文字と小文字を区別しています。したがって，環境に大文字で記述されている場合は，**名前**に大文字で指定する必要があります。

n には，環境に設定された項目の順番を 1 ～ 255 の範囲の整数値をもつ変数または数式で指定します。

名前を指定した場合，該当する項目に設定された値だけを返します。n を指定した場合，該当する項目名と共に返します。いずれの場合も，255 文字以内の文字ストリングの形で返されます。

名前と n のいずれの指定においても，該当する項目が見つからない場合は，空ストリングが返ります。

例： 現在，環境の中に次のものが設定されているとします。

```
COMSPEC=A:¥COMMAND.COM
PATH=A:¥;A:ASM;A:鈴木
```

次の ENVIRON$ 関数は項目名"COMSPEC"を指定して環境の内容を調べています。

```
PRINT ENVIRON$("COMSPEC")
A:¥COMMAND.COM
```

次の例は環境内の２番目の内容を調べています。

```
PRINT ENVIRON$(2)
PATH=A:¥;A:ASM;A:鈴木
```

EOF 関数

目的：　ファイルの終りを判別する値を返します。

構文：　―― EOF（ファイル番号）――

説明：　**ファイル番号**は，OPEN ステートメントでファイルに対して割り当てられた番号です。

この関数は，ディスクからの順次入力，または通信ファイルに対してのみ意味を持ちます。それ以外のファイルを指定しても無効です。また，入出力割り当てを行った場合，**ファイル番号**に 0 を指定すると標準入力装置のファイル終了状態を示します。

ディスクからの順次入力では，ファイルの終りに到達して次に読み込むべきデータが無くなった場合に-1（真），そうでない場合に 0（偽）が返ります。

通信ファイルでは，通信バッファー内の受理データが空である場合に-1（真），そうでない場合に 0（偽）が返ります。

一般にこの関数は，データの読み込み前にファイルの終りであるか否かを調べ，Input past end（入力が終りを越えた）エラーを防ぐために用います。

例：

```
10 OPEN "DATA" FOR INPUT AS#1
20 C=0
30 IF EOF(1) THEN END
40 INPUT #1,M(C)
50 C=C+1:GOTO 30
```

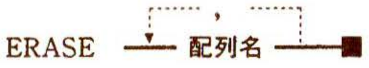
この例では，DATA というファイルから配列 M にデータを読み込み，ファイルの終りになるとプログラムを終了させています。

E

ERASE ステートメント

目的：　プログラムから指定された配列に割り当てられていた記憶域を消去します。

構文：



説明：　**配列名**には，不必要になった配列の名前を指定します。指定した**配列名**がプログラムに存在しない場合，Illegal function call（違法関数呼び出し）エラーが起きます。

ERASE は，指定された配列の記憶域の割り当てを解除します。実行中に大きな記憶域を必要とするプログラムでは，不必要となった配列を ERASE で削除しておくと有効です。また，配列の削除により同じ配列名を新たに宣言して使用することもできます（ただし，この場合配列の値は渡されません)。先に宣言した配列を削除せずに同じ名前を宣言すると，Duplicate Definition（二重定義）エラーが起こります。

すべての変数を消去するには CLEAR コマンドを使用します。

例：　次の例は配列領域を変えて記憶域の自由空間を調べています。FRE（"")は，その時点で使用可能な記憶域の大きさを返す関数です。結果は最初の記憶域により異なります。

```
10 START=FRE("")
20 DIM BIG(100,100):MIDDLE=FRE("")
30 ERASE BIG
40 DIM BIG(10,10):FINAL=FRE("")
50 PRINT START,MIDDLE,FINAL
Ok
RUN
 61993          21165          61474
Ok
```

ERDEV，ERDEV$ 変数

目的：

エラーが起こった装置名，およびエラー情報を返します。

構文：

— ERDEV —

— ERDEV$ —

説明：

ERDEV は，次の情報を含んだ 2 バイトの数値を返します。これらの情報の詳細については，☞ 日本語 DOS の内部構造とプログラム・インターフェース。

下位 8 ビット

DOS 割り込み INT24（致命的エラー処理ルーチン・ベクトル）で返されるエラー・コードの値

上位 8 ビット

装置ヘッダーの属性ビット（ビット 0～3，13～15）の値が返ります。

15	14	13		3	2	1	0

ERDEV$ は，エラーが起こった装置の名前を返します。文字装置（ディスプレイ，キーボード，またはプリンターの場合は 8 バイトの装置名が返り，ブロック装置（ディスク）の場合は 2 文字のドライブ指定子が返されます。

例：

```
10 ON ERROR GOTO 30
20 LOAD "A:PROG1":GOTO 50
30 PRINT (ERDEV MOD 256),INT(ERDEV/256),ERDEV$
```

このプログラム（一部分）をディスケット・ドライブＡのレバーをあげたままで実行すると，次の情報が表示されます。

```
2          0          A:
```

ERDEV を 256 で割った値が装置ヘッダーの情報，その余りがエラー・コードを示します。

ERR および ERL 変数

目的：

エラーが起こった場合，その行番号とエラー・コードを返します。

構文：

── ERR ──
── ERL ──

説明：

これらの変数は，エラーの種類に応じた処理を行うために，一般に，エラー処理ルーチン内で IF...THEN ステートメントと共に使用されます。これらの変数と関連深い「ON ERROR ステートメント」および「ERROR ステートメント」の記述も参照してください。

1 エラーが起こると，ERR 変数にエラー・コードが設定されます。BASIC が返すエラー・コードについては，☞ 文法編　付録 A「メッセージ集」

2 ERL 変数には，エラーの発生した行番号が設定されます。

エラーの起こった行番号と，特定の行番号を示す定数を，IF...THEN ステートメントで比較する場合，「ERL と定数を等号のどちら側に置くか」に注意する必要があります。

等号の右に定数を置いた場合，RENUM で行番号を付け変えると，この定数の値も自動的に新しい行番号に置き換えられます。プログラム行と比較する場合は，この形で指定します。

等号の左に定数を置いた場合，RENUM で行番号を付け替えても，定数の値は変わりません。直接モード入力によるエラーを調べる場合，RENUM の影響を受けないように定数を左側に置いて，「IF 65535=ERL THEN...」の形で比較します（直接モード入力に対する ERL の値は，常に 65535 です）。

E

例：

次のプログラムは，エラー・コード 53,62,71 のエラーに対して，それぞれの割り込み処理をしています。

```
100 MSG53A$="ファイルがありません。"
110 MSG53B$="ドライブA:に SAMPLE.DAT を入れてください。"
120 MSG62$="ファイルをすべて読み終えました。"
130 MSG71A$="ドライブA:のドアが開いています。"
140 MSG71B$="レバーを倒してください。"
150 ON ERROR GOTO 180
160 OPEN "A:SAMPLE.DAT" FOR INPUT AS#1
170 PRINT INPUT$(1,#1);:GOTO 170
180 IF ERR=53 THEN PRINT MSG53A$;MSG53B$:GOTO 210
190 IF ERR=62 THEN PRINT "SAMPLE.DAT ";MSG62$:GOTO 210
200 IF ERR=71 THEN PRINT MSG71A$;MSG71B$:GOTO 220
210 END
220 BEEP
230 INPUT "準備ができたら改行キー!",DUMMY$
240 RESUME
Ok
```

ERROR ステートメント

目的： BASIC エラーを模倣したりユーザー独自のエラーを発生させるために，エラーの状況を設定します。

構文： ERROR — エラー・コード ——■

説明： **エラー・コード**には，エラー発生と同時に ERR 変数に設定される値を，0 ～ 255 の範囲の整数式で指定します。

エラー・コードの値が BASIC のエラー・コードと同じ場合，このステートメントによって BASIC エラーの発生状態が模倣されます。ON ERROR ステートメントでエラー処理ルーチンが定義されている場合は，そのルーチンが実行されます。また，エラー・ルーチンが無い場合は，BASIC エラー・メッセージが表示された後，実行は停止します。

この機能は，ある状態（BASIC エラーでない状態，または別な BASIC エラーの状態）において，特定の BASIC エラーと同じ扱いをするために利用できます。

エラー・コードの値が BASIC のエラー・コードと異なる場合，ユーザー独自のエラーが発生します。独自のエラー・コードに対するエラー処理ルーチンを ON ERROR ステートメントで定義している場合は，そのルーチンが実行されます。エラー・ルーチンが無い場合，Unprintable error（表示不可能エラー）が表示された後，実行は停止します。

独自の**エラー・コード**には，BASIC エラー・コード（巻末のエラー・メッセージ一覧表を参照してください）と混同しないように，できる限り大きい数値（200 以上 255 以内）を使用することをお勧めします。

E

例：

1 次の例は，String too long（ストリングが長すぎる）というエラー・コード 15 の BASIC のエラーを模倣しています。

```
10 T=15
20 ERROR T
RUN
String too long in 20
Ok
```

2 次の例は，最適な在庫数量を求める適用業務プログラムの一部です。BASIC エラー・コードとして使用されていない値 210 を，ユーザー独自のエラー・コードに使用しています。

```
110 ON ERROR GOTO 400
120 INPUT "仕入数量は?";B
130 IF B>5000 THEN ERROR 210
・
・
・
400 IF ERR = 210 THEN PRINT "最大数量は5000を超えます"
410 IF ERL = 130 THEN RESUME 120
```

EXP 関数

目的：

指数関数を計算します。

構文：

―― EXP (x) ――

説明：

x は，88.02968 以下の値の数式で指定します。x がこの値を超えると，Overflow（桁あふれ）エラーが起こります。

この関数は，自然対数の底 e の x 乗を計算して返します。返される値は単精度で計算した値です。BASIC を始動するときに/D を指定すると，倍精度で計算した値を返します。

例：

```
10 X=2
20 PRINT EXP(X-1)
RUN
 2.718282
Ok
```

この例では，e の 1 乗，つまり e の値を求めています。

E

EXTERR (n) 関数

目的：　DOSの拡張エラー情報を返します。

構文：　EXTERR──(n)──■

説明：　nには0～3の値を次の求める情報に従って指定します。

n	返る情報（次の意味を持つ番号）
0	エラー・コード
1	エラー・クラス
2	対処（エラー処置のための指示）
3	場所（エラーが起きた場所）

EXTERRはDOS機能を使用したときの拡張エラー情報を返します。nは求めるエラー情報を識別する0～3の値を指定します。

BASICはDOS機能が使用されるたびにDOSの拡張エラー・コードを保持します。拡張エラー情報は最新の情報のみが返ります。

☞ 拡張エラー情報のエラー・コードの意味は DOSの内部構造とプログラム・インターフェース。

例：　次の例は，Aドライブに入っている書き込み防止されたディスケットに，プログラムを保管した場合生じる，エラーの状況を調べています。

```
10 ON ERROR GOTO 40
20 SAVE "A:EXAMPLE.BAS"
30 END
40 '--- エラー状況 ---
50 PRINT "ERR=";ERR
60 FOR X=0 TO 3
70 PRINT "EXTERR(";X;")=";EXTERR(X)
80 NEXT
90 RESUME NEXT
RUN
ERR=70
EXTERR(0)=19
EXTERR(1)=11
EXTERR(2)=7
EXTERR(3)=2
```

この例のエラー・コードは次の情報を提供しています。

ERR=70
　Permission Denied（書き込み要求拒否）

EXTERR(0)=19
　書き込み防止されているディスケットに書き込もうとした。

EXTERR(1)=11
　対象となる媒体に不備がある。

EXTERR(2)=7
　必要な処置を施し再び試みよ。

EXTERR(3)=2
　ブロック装置（ディスク）に問題あり。

FIELD ステートメント

目的：

ランダム・ファイル・バッファー内に，変数の領域を割り振ります。

構文：

FIELD [#] **ファイル番号** ，**幅** AS **ストリング変数** [， **幅** AS **ストリング変数** …]

説明：

ファイル番号には，ファイルのオープンに使用した番号を指定します。

幅は，変数に割り振る領域の大きさをバイト数で表したもので，数式で指定します。1つの FIELD ステートメントで割り振る変数の**幅**の合計は，OPEN ステートメントに指定したレコード長を超えることはできません。これを超えると，Field overflow（フィールドの桁あふれ）エラーが起こります。

ストリング変数には，ランダム・ファイルのアクセスに使用されるストリング型の変数の名前を指定します。

FIELD ステートメントは，ランダム・ファイル・バッファー内の各入出力データの領域に，プログラム内で使用する変数名を定義するために使用します（データの移動は行いません）。

ファイルとバッファーの間でのデータ入出力には，GET および PUT ステートメントを使用します。また，**ストリング変数**で指定した変数に値を代入するためには，LSET または RSET ステートメントを使用します。

注） **ストリング変数**への値の設定に，LET や INPUT ステートメントを使用しないでください。これにより，この変数は一般の変数とみなされ，バッファーへの定義付けが無効になります。

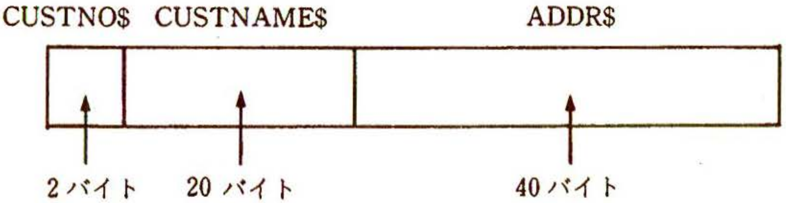
同じファイルに複数のFIELDステートメントを指定することも可能です。各FIELDは，それぞれバッファーの最初の桁から変数への割当てを行い，すべての割り当てが同時に有効に働きます。したがって，同じファイルから様式の異なるレコードが読み込まれる場合も，それぞれの様式に合う変数を使って処理することができます。ランダム・ファイルの詳細については，
☞解説編　付録A「ディスク入出力」。

例：

```
10 OPEN "A:CUST" AS #1
20 FIELD 1, 2 AS CUSTNO$, 20 AS CUSTNAME$, 40 AS ADDR$
30 LSET CUSTNAME$="山田一郎"
40 LSET ADDR$="東京都港区六本木"
50 LSET CUSTNO$=MKI$(7850)
60 PUT 1,1
70 GET 1,1
80 CNUM%=CVI(CUSTNO$):N$=CUSTNAME$
90 PRINT CNUM%,N$,ADDR$
RUN
 7850          山田一郎
東京都港区六本木
Ok
```

このプログラムは，顧客情報をもつランダム・ファイル（ファイル名「CUST」）に対する入出力を例示しています。

バッファーの領域は，次のように変数に割り当てられます。



行 30 ～ 50 で CUSTNO$（顧客番号），CUSTNAME$（顧客名），および ADDR$（住所）を使ってバッファーにデータを設定し，行 60 の PUT でファイルに書き出しています。さらに，行 70 でこのデータをファイルから読み込み，行 90 で表示しています。行 50 および行 80 の数値データの扱い方に注意してください。

次の例は，複数の FIELD ステートメントでランダム・ファイルバッファーを割り振ったものです。行 30 は，行 20 で定義したフィールドを失なうことなく次のフィールドをアクセスするための指定をしています。

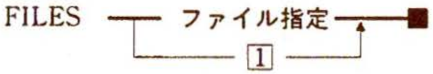
```
10 OPEN "FT.DAT" AS #1
20 FIELD #1, 10 AS A$, 20 AS B$
30 FIELD #1, 30 AS DUMMY$, 50 AS C$
```

FILES コマンド

目的：

ディレクトリー情報（ディレクトリー名，ファイル名，および残された記憶容量）を表示します。

構文：



[1] DOS 省略時ドライブの現行ディレクトリーにある全ファイルを表示

説明：

ファイル指定には，「装置名：」または「ドライブ指定子：ファイル名」形式のストリング式で，ファイルを指定します（ファイル名については，☞解説編　第3章「ファイル指定とパス」)。**ファイル指定**を省略すると，現行ディレクトリーに含まれるすべてのファイル名が表示されます。

装置名またはドライブ指定子を省略すると，省略時ドライブの現行ディレクトリーが参照されます。パス名を指定することもできます。

また，装置名またはドライブ指定子だけを指定すると，そこにあるすべてのファイル名が表示されます。

ファイル名中の文字には，疑問符（?）やアステリスク（＊）を使用することができます。「?」は，その1桁がどの文字（空ストリングも含む）にも対応することを示します。「＊」は，その部分がどの文字にも，何桁でも対応することを示します。これらの記号を使ってファイル名が指定された場合，対応するすべてのファイル名が表示されます。

ファイル指定に間違いがある場合，Bad file name（ファイル名の誤り）エラーが起こります。また，指定したファイルが存在しなかった場合，File not found（ファイルが見つからない）と表示されます。

例：

FILES	現行ディレクトリー内のすべてのファイル名を表示する。
FILES "*.BAS"	現行ディレクトリー内の，エクステンション BAS のすべてのファイル名を表示する。
FILES "B:*.*"	ドライブ B の省略時のパスに含まれるすべてのファイル名を表示する。
FILES "B:"	上の"B:*.*"と同様。
FILES "TEST??.BAS"	現行ディレクトリー内の，上 4 桁 「TEST」で次に 0 ～ 2 文字があり，エクステンションが BAS のファイル名すべてを表示する。
FILES "LVL1¥"	現行ディレクトリーのサブ・ディレクトリー「LVL1」のすべてのファイル名を 表示する。

FIX 関数

目的：　数式を切り捨てにより整数に変換します。

構文：　―― FIX (x) ――

説明：　x には数式を指定します。

この関数は，数式の値の小数点以下を切り捨てます。

数式を整数に変換する INT 関数との違いは，負数の変換において，INT が数式の値以下の負の整数を返すのに対して，FIX が数式の値以上の負の整数を返す点にあります。

INT 関数，および CINT 関数（四捨五入による整数変換）の記述も参照してください。

例：

```
PRINT FIX(45.67)
 45
Ok
PRINT FIX(-2.89),INT(-2.89),CINT(-2.89)
-2            -3            -3
Ok
PRINT FIX(-2.19),INT(-2.19),CINT(-2.19)
-2            -3            -2
Ok
```

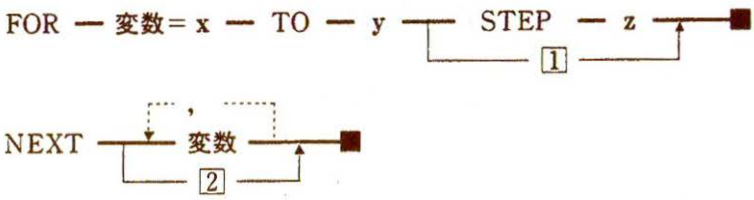
F

FOR および NEXT ステートメント

目的：

一連の命令を，必要な回数だけ繰り返し実行します。

構文：



1 STEP 1
2 最も新しい FOR に対する変数

説明：

FOR ステートメントは繰り返す一連の命令の最初に，NEXT ステートメントは一連の命令の最後に記述します（FOR と NEXT で繰り返される処理を FOR...NEXT ループと呼びます)。

FOR の**変数**は，「繰り返しが何度目であるか」を示すための変数（ループ変数と呼ぶ）の名前で，整数型または単精度型の変数を指定します（整数型の方が処理が速くなります)。

x は**変数**の**初期値**，**y** は**変数**の**最終値**，**z** は繰り返しのたびに**変数**に加算（負数であれば減算）していく**増分**です。これらは，いずれも数式で指定します。「STEP　z」を省略すると，増分が 1 とみなされます。

NEXT の**変数**は，対応する FOR の**変数**と同じものを指定します。ただし，NEXT は複数の FOR と対応させることもできるため，カンマ（,）で区切ってそれぞれの変数を指定することができます（詳細については，次ページの「ネストしたループ」に記述があります)。NEXT の**変数**を省略すると，最も新しい FOR に対応する変数が仮定されます。

FOR...NEXT ループは，FOR ステートメントでループ変数が検査され，指定範囲内の値であれば実行されます。

繰り返しごとに**増分**だけ加算（または減算）された**変数**の値が**最終値**より大きく（または小さく）なると，NEXT の次の処理に移ります（したがって，繰り返しの回数はこれら 3 つの値により決まります）。これらの値の指定には，次の 3 点に注意してください。

- 次の場合，FOR...NEXT ループは 1 度も実行されません。

  初期値>**最終値**　かつ　**増分**>0
  初期値<**最終値**　かつ　**増分**<0

- **初期値** = **最終値**の場合，**増分**が 0 でない限り，FOR...NEXT ループは 1 度だけ実行されます。

- **増分** =0 の場合，FOR...NEXT ループを無限に繰り返します（ループ変数の値にかかわりなく繰り返します）。したがって，何らかの条件によりループ外に分岐させるなどの方法を講じる必要があります。

ネストしたループ

FOR...NEXT ループはネストさせる（入れ子にする）ことができます。つまり，1 つの FOR...NEXT ループ内に，別な FOR...NEXT ループを含むことができます。ただし，内側のループが完全に外側のループに含まれている必要があります。

正しいネスト

```
FOR ステートメント 1
FOR ステートメント 2


NEXT ステートメント 2
NEXT ステートメント 1
```

間違ったネスト

```
FOR ステートメント 1
FOR ステートメント 2


NEXT ステートメント 1
NEXT ステートメント 2
```

ネストしたループには，それぞれ別々にループ変数を使用しなければなりません。また，各ループ変数の混同を防ぐために，ネストしたループの NEXT ステートメントには，必ず変数を指定してください。

ネストしたループの終了点が同じ場合，複数の NEXT ステートメントにまとめて指定することができます。これには，NEXT ステートメントに，各ループ変数をカンマ（,）で区切って続けて指定します。ループ変数の指定は，内側のループから順に行います。次の 2 つの形式は，同じネストを表します。

```
FOR 変数 1                FOR 変数 1
FOR 変数 2                FOR 変数 2
FOR 変数 3                FOR 変数 3

  NEXT 変数3              NEXT 変数 3，変数 2，変数 1
  NEXT 変数2
  NEXT 変数1
```

NEXT が FOR と対応しない場合，NEXT without FOR（FOR のない NEXT）エラーが起こります。

例：

1

```
10 J=10:K=30
20 FOR I = 1 TO J STEP 2  '--- 増分は2
30   PRINT I;
40   K=K+10:PRINT K
50 NEXT I
RUN
 1  40
 3  50
 5  60
 7  70
 9  80
Ok
```

2

```
10 J=0
20 FOR I=1 TO J
30   PRINT I
40 NEXT I
RUN
Ok
```

この例では，ループの初期値が最終値より大きく，増分が正の数（1）であるため，1度も実行されませんでした。

3

```
10 I=5
20 FOR I=1 TO I+5
30 PRINT I;
40 NEXT I
RUN
 1  2  3  4  5  6  7  8  9  10
Ok
```

このBASICは，ループ変数に初期値を設定する前に，最終値を求めます。最終値I＋5は10となり，ループは10回実行されます。

注） 他のBASICでは，ループ変数に初期値を設定した後，最終値を求めるものもあります。その場合，上記のプログラムでループは6回実行されます。

FRE 関数

目的：

BASIC が使用可能な記憶域（作業空間）の内，現在使用中でない領域の大きさをバイト数で返します。

構文：

── **FRE（仮引数）** ──

説明：

仮引数には，任意の数字またはストリングを指定します。

この関数は，未使用領域の大きさを返す前に，記憶域内のデータの圧縮を行います。BASIC のストリングは可変長（代入される値により長さが変わる）として扱われるため，値の長さを変えると無駄な空間が生じることがあります。ここでの圧縮とは，このような無駄な空間を詰め，使用可能な空間として解放することを示します。

BASIC が使用可能な空間を使い切った場合も，同様の圧縮処理が自動的に行われます。

圧縮には多少時間がかかるため，プログラム内での FRE 関数の多用は避けるべきです。

BASIC の作業空間の最大バイト数は，CLEAR ステートメントで設定します。
FRE が返す値は，この最大バイト数から，現在使用中の領域として予約されている領域（2.5K ～ 4K バイト）を引いたものになります。

例：

```
PRINT FRE(0)
 61684
Ok
```

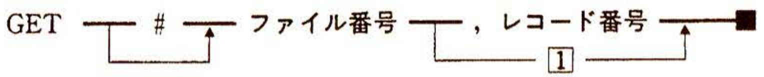
注） 各コンピューターの使用状況により，この例とは異なる値が返されます。

GET ステートメント（ファイル）

目的：

ランダム・ファイルからランダム・ファイル・バッファーへ，レコードを読み込みます。また，通信ファイルから通信バッファーへの読み込みも行います。

構文：



1 次のレコード

説明：

ファイル番号には，ファイルのオープン時に指定した番号と同じものを指定します。

ランダム・ファイルの場合，**レコード番号**には，読み込むレコードの番号を1～4,294,967,295の範囲の数式で指定します。**レコード番号**を省略すると，最後に読み込んだレコードの次のレコードを読み込みます。

通信ファイルの場合，**レコード番号**には通信バッファーから読み取るデータのバイト数を指定します。この値は，OPEN" COM ステートメントのLEN指定を超えない範囲で指定します。

GET ステートメントでバッファーに読み込んだレコードをプログラム内で使用するためには，INPUT#，LINE INPUT#，または FIELD ステートメントを使用します。詳細については，☞ 解説編 付録A「ディスク入出力」。

セクターの512バイトを有効に利用するために，DOSはブロック単位でレコードを読み込みます。このため，1回のGETステートメントの実行で，指定したレコード以外のレコードもバッファー内に読み込まれることがあります。GETで指定したレコードが，それまでのGETによりバッファー内に読み込まれていた場合，物理的な読み込みは行われません。

例：

```
10 OPEN "A:CUST" AS#1
20 FIELD 1,30 AS CUSTNAME$,30 AS ADDR$,35 AS CITY$
30 GET 1
40 PRINT CUSTNAME$,ADDR$,CITY$
```

この例では，ランダム・ファイル「CUST」行 30 でレコードを読み込んでいます。バッファー内に読み込まれたレコードは，行 20 で定義された変数名を使って参照されます。

GET ステートメント（グラフィック）

目的：

画面上の一定区域内の点（ドット）に関する情報を読み取ります。

構文：

GET — (x1, y1) − (x2, y2), **配列名** ——■

説明：

(x1, y1), (x2, y2) は，画面上の範囲を決める座標を，絶対座標で指定します。この 2 点を対角の頂点とする四角形が，情報を読み取る範囲となります。

配列名は，読み取った情報を入れる配列の名前で，数値配列を指定します。

範囲内の画面イメージを保持するために，この配列全体で次の計算式から求められるバイト数が必要です。

単色と 4 色カラーのとき

4 + INT ((x*n + 7) /8) *y

16 色カラーのとき

4 + INT ((x + 7) /8) *n*y

x： 範囲内の x 座標の点の数
y： 範囲内の y 座標の点の数
n： 画素ビット数

画面	n
単色……	1
4 色カラー……	2
16 色カラー……	4

G

したがって，配列を定義する際（DIM ステートメント）の最大添字は，上記の式から求められた全体のバイト数を，次の各要素のバイト数で割った値となります。

- 整数配列の場合……各 2 バイト
- 単精度配列の場合…各 4 バイト
- 倍精度配列の場合…各 8 バイト

配列に入る情報は，最初の 2 バイトが x 方向の範囲の大きさ，次の 2 バイトが y 方向の範囲の大きさ，残りのバイトが画面上の各点のデータです。このバイト構成は，配列が整数，単精度，倍精度のいずれの型であっても同じです。

GET で読み取った情報は，PUT ステートメント（グラフィック）により別な位置に書き出すことにより，図形の高速移動に使用することができます。情報はそのままの形で GET から PUT へ渡されるため，個々の情報を解析する必要はありません。

例：

16 色カラー・グラフィック・モードで，横 20 ドット，縦 20 ドットの画面イメージは，全体で 4 + INT ((20 + 7) /8) ＊4＊20，つまり 244 バイトの情報をもちます。これを整数配列に記憶する場合 244/2，すなわち 122 個の配列要素が必要です。

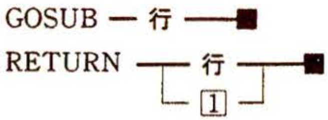
```
10 SCREEN 2,1:LOCATE,,0:DIM C%(122)
20 CIRCLE (10,10),10,7
30 PAINT (10,10),6,7
40 GET (1,1)-(20,20),C%:PUT (0,0),C%
50 FOR I=0 TO 280 STEP 7
60   PUT (I,I),C%
70   PUT (I,I),C%
80 NEXT I
90 LOCATE,,1
```

GOSUB および RETURN ステートメント

目的：

サブルーチンに分岐して実行し，また，分岐した箇所に戻って実行を続けます。

構文：



[1] GOSUB の次の文

説明：

GOSUB ステートメントでサブルーチンに移り，RETURN ステートメントで戻ります。

行は，サブルーチンの最初の行番号を指定します。

GOSUB によるサブルーチンの実行は，プログラム中で何度でも使用できます。また，サブルーチン内で別なサブルーチンを実行させることもできます（ただし，サブルーチンのネストの多重度は，使用可能な記憶域の大きさにより制限されます）。

サブルーチン内に指定された RETURN により，最後に実行された GOSUB の次のステートメントから実行が続けられます。サブルーチン内に複数の RETURN を指定して，条件により異なる出口を使うことも可能です。

プログラムが偶然にサブルーチンに入ることを防ぐために，サブルーチンの直前には STOP，END，または GOTO を置くようにしてください。

条件により異なるサブルーチンに分岐する場合，ON...GOSUB ステートメントを使用します。

例：

```
10 PRINT "サブルーチンを呼ぶ"
20 GOSUB 50
30 PRINT "サブルーチンから戻る"
40 END
50 PRINT "!!! サブルーチン内 !!!"
60 RETURN
RUN
サブルーチンを呼ぶ
!!! サブルーチン内 !!!
サブルーチンから戻る
Ok
```

この例では，行 10 が実行された後，行 50 から始まるサブルーチンが実行されます。さらに行 60 の RETURN により分岐元の次の行，つまり行 30 から実行されます。さらに，このままサブルーチンが再実行されることを防ぐため，行 40 に END を指定しています。

GOTO ステートメント

目的： 通常のプログラムの実行順序を変え，指定した行番号へ無条件に分岐します。

構文： GOTO ― 行 ―■

説明： 行には，分岐先の行番号を指定します。

分岐先が実行可能なステートメントである場合，その行から実行が継続されます。また，分岐先行が REM や DATA などの非実行ステートメントである場合，それ以降の最初の実行可能ステートメントから続行されます。

直接モードでの GOTO は，プログラム中の任意の行をやり直すデバッグ用に利用できます。

条件により分岐先が異なる場合，ON...GOTO を使用します。

例：

```
10 DATA 5,7,9
20 READ R
30 PRINT "半径：";R;"面積=";3.14*R^2
40 GOTO 20
RUN
半径： 5 面積= 78.5
半径： 7 面積=153.86
半径： 9 面積=254.34
Out of DATA in 20
Ok
```

HEX$ 関数

目的： 10 進の数値を，16 進数（ストリング）に変換します。

構文： —— HEX$ (**n**) ——

説明： n には，-32768 ～ 65535 の範囲の数式を指定します。

n が負数の場合，2 の補数形式で表されます（つまり，-n の 16 進表示は，65536-n の 16 進表示と等しくなります）。

10 進の数値を 8 進数（ストリング）に変換するためには，OCT$ 関数を使用します。

例： 次のプログラムは，入力された 10 進数を 16 進表示します。

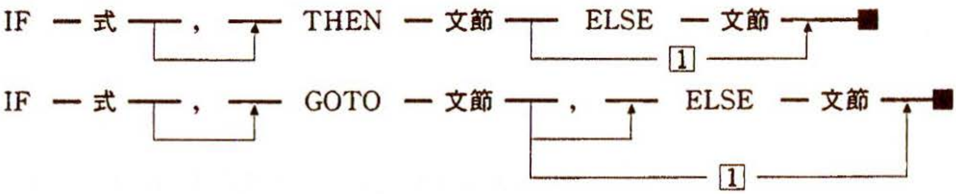
```
10 INPUT X
20 A$=HEX$(X)
30 PRINT "10進数" X "の16進表示は"A$
RUN
? 32
10進数 32 の16進表示は20
Ok
RUN
? 1023
10進数 1023 の16進表示は3FF
Ok
```

IF ステートメント

目的：

式の結果にしたがって，プログラムの流れを判断をします。

構文：



1 式が偽のときは何も実行せずに次のステートメントへ行く

説明：

式には，判断のための条件を数式で指定します。

文節には，1つのステートメント，またはコロン（:）で区切って並べた一連のステートメントを指定します。

文節に GOTO ステートメントを1つだけ指定する場合は，省略形として THEN または ELSE の直後に分岐先の行番号だけを指定することができます。また，THEN の文節に GOTO ステートメントを1つだけ指定する場合は，THEN を省略して GOTO から記述することもできます。

式の値が真（式の条件が満たされたとき）の場合，THEN 側に指定された**文節**が実行され，ELSE 側の**文節**は無視されます。逆に**式**の値が偽（式の条件が満たされないとき）の場合，THEN 側の**文節**は無視され，ELSE 側の**文節**が実行されます（ELSE が指定されていなければ，何も実行されません）。いずれの場合も，**文節**に分岐命令が含まれない限り，**文節**の実行後は IF ステートメントの次から実行されます。

分岐先の行番号が存在しない場合，Undefined line number（行番号が不定）エラーとなります。

IF の使用について，次の3点に注意してください。

- 式で単精度または倍精度の値を特定の値と比較する場合，等号（=）でなく範囲（＜＞）を利用する。

  単精度および倍精度の数値は，内部的に浮動小数点 2 進数形式で記憶されるため，厳密には正しい値をもたない場合があります（近似値をもちます）したがって，特定の値と等しいか否かを比較する場合，その値との差がどれだけあるかで判断してください（例 3 を参照)。

- IF...THEN...ELSE は 1 ステートメントとして指定する。

  THEN および ELSE とその文節は，IF と同じステートメントとして指定する必要があります。したがって，[Ctrl] +[↵] により改行された場合を除き（ステートメントの継続とみなされる)，途中で改行したり，別な行番号を付けてはいけません。

- IF ステートメントのネストでは，ELSE に 1 番近い THEN が対応するものとみなされる。

  IF ステートメントは，1 行の長さを限度としてネストさせることが可能です。ELSE 指定が無い IF の影響で THEN（または THEN に代わる GOTO）と ELSE の数が一致しない場合，1 番近くにある THEN（または GOTO）と ELSE から順に，対応するものとみなされます（例 4 を参照)。

例：

1

```
120 IF A=B THEN IF B=C THEN PRINT "A=C"
            ELSE PRINT "A<>C"
```

この例で，ELSE は「THEN PRINT A=C」に対応するものとみなされ，「A が B と等しく，かつ B は C と等しくない」場合のみ実行されます。A=B に対する偽の処理は何も行われません。

2

```
100 IF (I>10) AND (I<20) THEN DB=1986-1:GOTO 300
             ELSE PRINT "範囲外"
```

この例では，I が 10～20 の間にある場合，DB に値を設定して行 300 に分岐します。I の値がそれ以外の場合，範囲外を表示して次のステートメントに進みます（GOTO 300 と ELSE の間は，[挿入]＋[Ctrl]＋[↵]で改行しています）。

3

```
500 IF ABS(A-1.0)<1.0E-6 GOTO 100
```

ここでは，単精度型，または倍精度型をもつ変数と特定の値を比較する方法を例示しています。A の値を 1 と比較するために，A と 1 の差の絶対値を求め，それが 0.000001 より小さい場合に両者が等しいとみなしています。

4

```
200 IF I THEN GET #1,I
```

このステートメントは，I が 0 でない場合にレコードを I に読み込みます。I が 0 の場合は何も行いません。

5

```
300 IF IOFLAG THEN PRINT A$ ELSE LPRINT A$
```

このステートメントは，変数 IOFLAG の値が 0 の場合はプリンターに出力し，その他の場合は画面上に出力します。

INKEY$ 変数

目的：　キーボードから入力された文字を返します。

構文：　── INKEY$ ──

説明：　INKEY$ は，キーボード・バッファーに多くの文字が入っていても最初の1文字分だけが返ります。返るストリングは次のとおりです。☞ 解説編付録D「コード体系」。

- バッファー内に未処理の文字が無い場合…空ストリング（長さ0）
- 最初が1バイト・コード文字の場合… 1文字ストリング
- 最初が2バイト・コード文字の場合… 2文字ストリング
  （または，拡張コードの場合）

1度 INKEY$ により参照された文字は，バッファー内から削除されます。したがって，同じ入力文字を繰り返し参照する場合は，INKEY$ の値をストリング変数に保存した後，他のステートメントで使用してください。

INKEY$ の使用中に，キーボードから入力した文字は，画面表示されません。また，[↵] キーを押した場合は，INKEY$ に復帰文字が入ります。

次に挙げるキー入力は，INKEY$ では参照できません。

- 漢字モードのキー　漢字の選択または無変換が指示されるまでのキー入力は参照できません
- [Break]キー機能（プログラムの中断機能）
- [Pause]キー機能（システムの一時停止機能）
- [前面] ＋ [Ctrl] ＋ [削除] キー（システム・リセット機能）
- [印刷]キー（画面印刷機能）

例：

1　次のプログラム（一部分）は，何らかのキーが押されるまで，プログラムを中断させます。

```
110 PRINT "実行の再開には，何かキーを押してください"
120 A$=INKEY$:IF A$"" THEN 120
```

2　次の例は，全角文字（2バイト・コード文字）の入力だけを表示し，[↵]キーにより終了します。

```
10 KB$=INKEY$:IF KB$="" THEN 10
20 IF ASC(KB$)=13 THEN 60
30 IF LEN(KB$)=1  THEN 10
40 PRINT KB$;
50 GOTO 10
60 END
```

I

INP 関数

目的：　入力ポートの 1 バイトを返します。

構文：　—— INP (**n**) ——

説明：　n には，入力ポートアドレスを，0 ～ 65535 の数値で指定します。
この関数は，アセンブラーの IN 命令と同等の機能をもちます。

OUT ステートメントの記述も参照してください。

例：

```
10 C1=&H3FD
20 S%=INP(C1)
30 IF (S% AND 2)<>0 THEN PRINT "パリティー・エラー"
40 IF (S% AND 4)<>0 THEN PRINT "オーバーラン"
50 IF (S% AND 8)<>0 THEN PRINT "フレーム・エラー"
60 IF (S% AND 16)<>0 THEN PRINT "中断エラー"
```

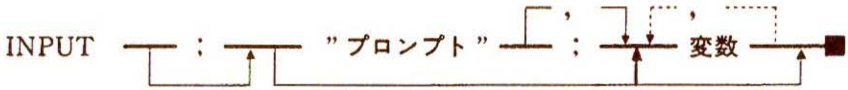
この例では，アドレス&H3FD のポートを検査して，回線状況を調べています。

INPUT ステートメント

目的：

プログラム実行中，キーボードからの入力を受け取ります。

構文：



説明：

INPUT に続く**セミコロン (;)** は，データ入力後のカーソル位置を決定します。**セミコロン**を省略すると，データ入力後にカーソルは次の行の先頭に移動します。セミコロンを指定すると，カーソルは元の位置から移動しません。

"プロンプト"には，プロンプト（キーボードからの入力を要求するメッセージ）に表示するストリング定数を指定します。**"プロンプト"**を省略すると，疑問符（?）1 つがプロンプトとして表示されます。

"プロンプト"の後の**カンマ (,)** および**セミコロン (;)** は，**"プロンプト"**に疑問符（?）を付加するか否かを決めます。**カンマ**を指定した場合，プロンプトの後には何も表示されません。**セミコロン**を指定した場合，プロンプトの最後に疑問符が表示されます。

変数には，キーボードから入力された値を入れるための変数名を指定します（配列要素も指定することができます）。

INPUT ステートメントは，画面にプロンプトを表示した後プログラムを停止させ，キーボードから入力があるまで実行を保留します。

入力データは，指定した変数に割り当てられます。**変数**をカンマ（,）で区切って複数並べた場合，入力データもカンマ（,）で区切って並べます。このとき，変数とデータの個数が一致し，さらにそれぞれの変数とデータの型が一致している必要があります。データ入力に関して，次の規則があります。

- ストリングのデータは，カンマや，先行ブランクまたは後続ブランクを含まない限り，引用符で囲む必要はありません。

- 単一の入力データ，または複数入力する場合の最終データは，単に ↵ キーを押してカンマを省略することができます（最終データの前のカンマは必要です）。この場合，対応する変数が数値変数であれば 0 が，ストリング変数であれば空ストリングが設定されます。

指定した変数と入力データの数や型が一致しない場合，Redo from start（始めからやり直し）というメッセージが表示されます。複数のデータ入力時に，最終データ以前に ↵ キーを押した場合も同様にやり直しとなります。入力のやり直しが指示された場合，一部の正しく入力されたデータも，変数には代入されません。

例：

```
10 INPUT "END";X$:IF X$="END" GOTO 40
20 INPUT;A:PRINT "*2 =" A*2
30 GOTO 10
40 END
RUN
END?
```
（↵ キーを押すと次の行に?が表示されます）
```
? 5
```
（数値 5 を入力した後 ↵ キーを押します。
```
? 5*2 = 10
```
（入力した 5 に続けて答えが表示されています）
```
END? END
```
（ここで END を入力して終了させます）

この例は，END?に対して END と入力するまで，入力数値の 2 倍を計算します。行 10 と行 20 の INPUT の働きについて，次の 2 点に注目してください。

- 行 10 はプロンプト（疑問符付き）を表示し，行 20 は疑問符しか表示しない。

- データ入力後，行 10 ではカーソルが次の行の先頭に移動し，行 20 ではそのままの位置にある（行 20 の PRINT による画面表示が入力データに続いて出される）。

次のプログラムは簡単な横棒グラフを表示した例です。

```
10 SCREEN 0,1
20 LOCATE 1,1 : PRINT"---横棒グラフ---";
30 FOR I = 1 TO 7
40   ITEM$ = "項目"+STR$(I)
50   COLOR 7 : LOCATE 20,1 : INPUT "データは";INDATA
60   IF INDATA > 70 THEN 50
70   IF INDATA = 0 THEN COLOR 7,0,0 : END
80   LOCATE I*2+1 : PRINT ITEM$+" ";
90   COLOR I+16 : FOR J = 1 TO INDATA : PRINT " ";: NEXT
100 NEXT I
110 COLOR 7,0,0 : END
```

I

INPUT# ステートメント

目的：

順次データを読み込み，変数に割り当てます。

構文：

INPUT ― **#ファイル番号** ―, **変数** ―■

説明：

ファイル番号は，ファイルを入力用にオープンした際に指定した番号です。ディスク上の順次ファイルに限らず，通信カードやキーボードからの順次データ，およびランダム・ファイルからの順次入力にも使用できます。

変数には，入力された値を入れるための変数名を指定します（配列要素も指定することができます）。

ファイル内のデータと**変数**の型は，一致している必要があります。ファイル内のデータは，変数の型に従って，次の規則で読み込まれます。

変数がストリング型の場合

先行スペース，改行文字，および復帰文字は無視され，それ以外の最初の文字から読み取られます。最初の文字が引用符（"）であれば「引用符付きデータ」と判断され，次の引用符との間にある文字が**変数**に割り当てられます。また，最初の文字が引用符でなければ「引用符なしデータ」と判断され，カンマ，改行文字，または復帰文字が現れるまでのすべての文字が**変数**に割り当てられます（ただし，255 文字を超える場合は 255 文字目までがデータとみなされます）。

したがって，データとしてカンマ，改行文字，または復帰文字を含む場合，データの両端に引用符（"）が必要です（この場合引用符付きストリング内にデータとして引用符を含むことはできません）。ストリング内にカンマ，復帰文字，および改行文字を含まない場合，両端の引用符は不要です。

変数が数値型の場合

先行スペース，改行文字，および復帰文字は無視され，それ以外の最初の文字から読み取られます。スペース，カンマ，改行文字，または復帰文字が現れるまでのすべての文字が，データとして**変数**に割り当てられます。

一般に，通信データですべての文字に意味を持たせる場合は，区切り文字を無視するINPUT#ステートメントよりも，INPUT$ 関数の使用をお勧めします。通信データの詳細については，☞ 解説編 付録C 「通信」。

例：

```
100 '--- 順次ファイルの入出力 ---
110 '
120 '--- データ入力
130 SCREEN 0,0
140 OPEN "SAMP.DAT" FOR OUTPUT AS #1
150 INPUT "データは";INDATA$
160 IF INDATA$ = ""THEN CLOSE : GOTO 190
170 PRINT #1,INDATA$
180 GOTO 150
190 '--- データ表示
200 OPEN "SAMP.DAT" FOR INPUT AS #2
210 INPUT #2,OUTDATA$
220 PRINT OUTDATA$
230 IF EOF(2) THEN END
240 GOTO 210
```

```
RUN
```

```
データは?時田奈保子
データは?庄司あきみ
データは?篠原武志
データは?
時田奈保子
庄司あきみ
篠原武志
```

I

INPUT$ 関数

目的：

キーボードなどのファイルから読み取った文字を任意のバイト数だけ返します。

構文：

— INPUT$ （— n —， — # — ファイル番号 — ）

説明：

nには，ファイルから読み込む文字のバイト数を，数式で指定します。

ファイル番号は，ファイルのオープン時に指定した番号です。**ファイル番号**を省略すると，キーボードから入力したデータが返されます。

キーボード入力を INPUT$ で読み取る場合，n バイトを読み取るまでは，次のものを除くすべてのキー入力が，INPUT$ に保持されます（この間の入力文字は画面に反映されません）。

- 漢字モードのキー　漢字の選択，または無変換を指示するまで無効
- [Break] キー（INPUT$ の読み取りを中断する機能）
- [Pause] キー（プログラムの一時停止機能）
- [前面] ＋ [Ctrl] ＋ [削除] キー（システム・リセット機能）
- [印刷] キー（画面印刷機能）

INPUT$ 関数をキーボード入力に使用する場合，読み取るデータの長さが指定できる点を除くと，INKEY$ と同じ機能をもちます（次に記述している INPUT¥ 関数も同様です）。これらの関数は，特に制御文字の入力（[↵] キーや [後退] キーなど）を読み取る場合に役立ちます。

漢字まじりのデータが入力される場合，読み込むデータをバイト数で指定するINPUT$ より，文字数で指定できる INPUT¥ 関数の方が有効です。

例：

1 次の例は，順次ファイルの内容を 16 進数で表示します。

```
10 OPEN "DATA" FOR INPUT AS#1
20 IF EOF(1) THEN 50
30 PRINT HEX$(ASC(INPUT$(1,#1)));
40 GOTO 20
50 PRINT
60 END
```

2 次のプログラム（一部分）は，キーボードから入力される文字により，プログラムの流れを変えています。

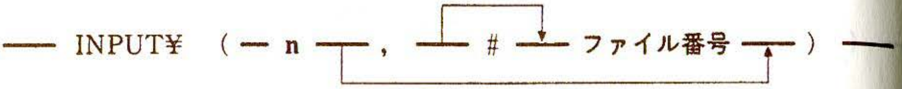
```
100 PRINT "G(継続)またはS(中止)を押してください"
110 X$=INPUT$(1)
120 IF X$="G" THEN 500
130 IF X$="S" THEN 700 ELSE 100
```

INPUT¥ 関数

目的：

キーボードなどのファイルから読み取った文字を，任意の文字数だけ返します。

構文：



説明：

nには，ファイルから読み込む文字数を数式で指定します。

ファイル番号は，ファイルのオープン時に指定した番号です。**ファイル番号**を省略すると，キーボードから入力したデータが返されます。

INPUT¥ 関数の働きは，次の点を除いて INPUT$ 関数と同様です(INPUT$ 関数の記述を参照してください)。

INPUT¥ は，1バイト・コード文字も2バイト・コード文字も，どちらも1文字として読み込みます。したがって，データ内に1バイト・コードと2バイト・コードが混在していても，希望する文字数を読み取ることができます。

例：

次のプログラムは，キーボードから2バイト・コード文字が入力された場合はその文字を表示し，[↵]キーが押された場合は終了します。

```
10 A$=INPUT¥(1)
20 IF ASC(A$)=13 THEN 60
30 IF ASC(A$)<256 THEN 10
40 PRINT A$;
50 GOTO 10
60 END
Ok
```

INSTR 関数

目的：

文字列を探索し，その中に含まれる特定の文字（または文字列）の位置を返します。

構文：

```
── INSTR （ ┬─ n, ─┬─ x$, y$ ─ ） ──
           └─ [1] ─┘
```

[1] 1

説明：

n は，x$ 中で y$ の探索を開始する位置を，先頭からのバイト数で示します。これは，1 ～ 255 の数式を使って指定します。n を省略すると，1 とみなされ先頭から探索されます。範囲外の値を指定すると，Illegal function call（違法関数呼出し）エラーが起こります。

x$ には探索される文字列を示し，y$ には見つけ出す文字列を示します。いずれもストリング変数，ストリング式，またはストリング定数で指定します。

x$ のバイト数より大きい n を指定した場合，または目的の y$ が x$ 内にない場合，INSTR は 0 を返します。また，y$ に空ストリングを指定した場合，INSTR は n（省略時は 1）を返します。

INSTR による探索はバイト単位で行われます。1 バイト・コード文字と，2 バイト・コード文字の 2 バイト目が同じコードである場合，両者の区別はつきません。したがって，y$ の 1 バイト・コード文字が x$ 中の 2 バイト・コード文字の 2 バイト目に一致すると，間違った位置が返される場合があり，注意が必要です。（例 2.および 3.を参照）

例：

1

```
10 A$="ABCDEB":B$="B"
20 PRINT INSTR(A$,B$);INSTR(4,A$,B$)
RUN
 2  6
Ok
```

この例は，ストリング ABCDEB の中で B を探索します。先頭から探索すると 2 バイト目に，4 バイト目から探索すると 6 バイト目に，それぞれ B が見つかります。

2

```
PRINT INSTR(1,"若葉［t］","t")
 4
Ok
```

これは，2 バイト・コード文字の 2 バイト目と 1 バイト・コード文字コードが同じため，間違った位置が返された例です（" 葉" の内部コードは 9774，" t" の内部コードは 74 です）。

3 次に，同じデータから正しい位置を求める例を挙げます。

```
10 X$="若葉［t］":Y$="t":N=1
20 L1=KLEN(X$):L2=KLEN(Y$)
30 FOR I=N TO L1-L2+1
40 P=KPOS(X$,I)
50 IF INSTR(P,X$,Y$)=0 GOTO 80
60 IF INSTR(P,X$,Y$)=P GOTO 80
70 NEXT I
80 PRINT P
90 END
RUN
 6
Ok
```

INT 関数

目的：　与えた数式の値以下で，最大の整数を返します。

構文：　—— INT (x) ——

説明：　x には，数式を指定します。

これは，他のプログラム言語でフロア関数と呼ばれることがあります。

この関数と同様に，数式から整数を返す FIX 関数，および CINT 関数の記述も参照してください。

例：

```
PRINT INT(45.67)
 45
Ok
PRINT INT(-2.89)
-3
Ok
```

数式の値が負の場合の結果に注意してください。

I

IOCTL ステートメント

目的：

ユーザーが作成した装置駆動ルーチン（文字装置）へ，制御データを渡します。

構文：

IOCTL ─┬─ # ─┬─ **ファイル番号，制御データ**
　　　　　└────┘

説明：

ファイル番号は，装置駆動ルーチンのオープンに使用したファイル番号を指定します。

制御データには，文字装置の装置駆動ルーチンに渡す制御データを，255文字以内のストリング式で指定します。制御データは，5～6文字の英数字で構成されたコマンドを，セミコロン（;）で区切って並べたものです。

ユーザーは，独自の入出力処理を行うために，装置駆動ルーチンを作成してシステムに組み込むことができます（装置駆動ルーチンについては，☞ DOSの内部構造とプログラム・インターフェース。標準のDOSの装置駆動ルーチンは，制御用のコマンドを処理しません。

指定したファイル上に装置駆動ルーチンがない場合は Illegal function call（違法関数呼出し）エラーが，指定されたファイルがオープンされていない場合は Bad file number（ファイル番号の誤り）エラーが，制御データの間違いがあった場合は Device fault（装置上の誤り）エラーが発生します。

例：

#1でオープンしたプリンターの装置駆動ルーチンに制御用コマンド PL66（ページ当たりの印刷行を66に設定）を送る場合，次のように指定します（コマンドはユーザーがルーチン内で任意に定義します）。

```
30 IOCTL #1,"PL66"
```

IOCTL$ 関数

目的：

ユーザーが作成した装置駆動ルーチン（文字装置）から，装置の状態を示すコードを返します。

構文：

— IOCTL$ （ [#] **ファイル番号**） —

説明：

ファイル番号は，装置駆動ルーチンのオープン時のファイル番号を指定します。

一般に，この関数は，IOCTL ステートメントで渡した制御データが正しく実行されたか否かを確認するために使用します。ただし，それ以外のときにこの関数を利用して，現在の装置駆動ルーチンの情報を得ることもできます。

指定したファイル上に装置駆動ルーチンがない場合は Illegal function call（違法関数呼出し）エラーが，指定されたファイルがオープンされていない場合は Bad file number（ファイル番号の誤り）エラーが，それぞれ発生します。

I

例：

次の例では，" PRINT1" という装置駆動ルーチンに対する制御データ PL66 が正しく実行されたか否かを調べ，正しく働かなかった場合にファイルをクローズします。

```
10 OPEN "PRINT1" AS #1
20 IOCTL #1,"PL66"
30 IF IOCTL$(1)="0" THEN CLOSE #1
```

JIS$ 関数

目的： ストリングの最初の文字の JIS16 進コードを，ストリング形式で返します。

構文： —— JIS$ (x$) ——

説明： x$ には，ストリング式を指定します。

この関数は，ストリングの最初が 1 バイト・コード文字の場合は 1 ～ 3 桁，2 バイト・コード文字の場合は 4 桁の JIS16 進コードを表すストリングを返します☞解説編　付録 D「コード体系」)。

x$ が空ストリングの場合（つまり x$ として何も指定しない場合）, Illegal function call（違法関数呼出し）エラーが起こります。

ストリングの最初の文字の連続コードを求める場合は，ASC 関数を使用します。

例： 次の例は，漢字「日」の JIS16 進コード（467C）を表示しています。ストリングの内容が異なっても，最初の文字が同じであれば，同じストリングが返されます。

```
10 X$="日本一":Y$="日付"
20 PRINT JIS$(X$),JIS$(Y$)
RUN
467C          467C
Ok
```

KEY ステートメント

目的：

任意の文字を表示・入力する機能，またはキー割り込みを発生させる機能を，特定のキーに割り当てます。また，PF キーに割り当てられている機能を表示します。

構文：

1 KEY ― n，x$ ―■

2 KEY ON ―■
KEY OFF ―■
KEY LIST ―■

説明：

1 この形式のときの KEY ステートメントは次の 2 種類の割り当てが可能です。

- 任意の文字（コマンドなど）を表示または入力する機能
- キー割り込みを発生させる機能

それぞれの機能を混用すると，Illegal function call（違法関数呼び出し）エラーとなります。

文字の表示・入力機能の割り当て

n はキーを識別する番号を 1 ～ 10，30，31 の範囲の数式で指定します。この値はそれぞれ PF1 ～ PF12 に対応します。

このときの x$ は，PF キーに対して割り当てる文字を，15 バイト以内のストリング式で指定します（定数で指定する場合は引用符が必要です）。

K

各 PF キーは，次のストリングが初期設定として設定されています。

PF1 LIST	PF2 RUN	PF3 LOAD"
PF4 SAVE"	PF5 CONT	PF6 ," LIST:"
PF7 TRON	PF8 TROFF	PF9 KEY
PF10 SCREEN 0, 0		

この機能を使用すると現在のストリングの設定を変更することにより，頻繁に使用するコマンドの入力や表示を，PF キー 1 つだけの操作に簡略化することができます。この PF キーの指定を無効にするためには，空ストリングを割り当てます。

INKEY$ は，PF キーのストリングの最初の 1 文字を返します。

キー割り込み発生機能の割り当て

n はキーを識別する番号を 15 ～ 25 の範囲の数式で指定します。この範囲は任意のキー割り込みを定義するために用意された番号です。ON KEY ステートメントや KEY (n) ステートメントではこのキーを使用して割り込み処理の設定を行います。

このときの x$ は，**CHR$ (シフト) + CHR$ (走査コード)** を使用して，割り込みを発生させるキーを指定します。独自のキーの割り込み処理はこの設定したキーの組み合わせに対して定義します。**CHR$ (シフト)** には次に示すキーの中の 1 つを対応する 16 進数で指定します。**走査コード**については本書の付録にある走査コードを参照してください。

シフト：シフト状況を示す次の **16 進数で**指定します。

CAPS	&H40
前面キー	&H08
Ctrl	&H04
⇧	&H01,&H02,&H03 のいずれか

CAPS キーをシフトしていない状態を設定するときは，&H0 を使用します。

⇧ キーの指定は，&H01，&H02，&H03 のいずれかを指定しますが，左右の区別はありません。3 つの値のいずれを使用しても ⇧ キーの割り込みを指定できます。

走査コード：&H01 ～&H7F の範囲の走査コードを 16 進数または 10 進数で指定します。

BASIC は次の規則に基いてキー割り込みの設定を調べ，処理します。

1 まず Ctrl ＋ 印刷 キー（エコー印刷）が設定されているか否か調べます。この組み合わせで割り込み処理するキーを設定してもその設定は無効となります。Ctrl ＋ 印刷 キーはどのような場合であってもエコー印刷として機能します。

2 次に PF キーまたはカーソル移動キー（←，→，↑，↓）が設定されているか否か調べます。これらのキーに割り込み処理を設定する場合には，ON KEY (n) ステートメントでキーを指定します。KEY ステートメントで指定しても無視されます。

3 最後に実際に定義されたキーに対する処理を行います。

K

注）● 割り込み処理されたキーはキーボード・バッファー内に保持されません。キーが押されたことを BASIC に知らせるだけです。

● Break キー機能や Ctrl + 前面キー + 削除 （システム・リセット）に対して割り込みを設定すると，プログラムを強制終了させる手段がなくなるため，注意が必要です。

2 この形式の KEY ステートメントは，PF キーに割り当てられている値の表示を制御します。

KEY ON は，画面の 24 行目に PF キーの値を PF1 ～ PF9 は 6 バイトずつ，PF10 は 5 バイト表示します。画面に表示できる文字数により，表示されるキーの数は異なります。BASIC の始動時には，**KEY ON** の状態になっています。

KEY OFF は 24 行目の PF キーの表示を消去します。この行はプログラムで使用できるようになります（キーの機能には影響しません）。

KEY LIST は，PF キーに割り当てられている文字をそれぞれ 15 バイトずつ表示します。

例：

1 PF1 キーに"FILES"+ ↵ の機能を割り当てます。

```
10 KEY 1,"FILES"+CHR$(13)
```

2 PF2 キーに割り当てられている文字を無効にします。

```
50 KEY 2,""
```

3 この例は大文字の P（CAPS + P ）に対して，割り込み処理を設定しています。KEY ステートメントは割り込みキーの設定をしています。ON KEY ステートメントは割り込み処理先を示しています。KEY（n） ON は割り込みの使用を有効に設定しています。

```
100 KEY 15,CHR$(&H40)+CHR$(&H21)
110 ON KEY(15) GOSUB 1000
120 KEY(15) ON
```

4 [Ctrl] + [⇧] + [A] の組み合わせを設定するには次のように行います。[Ctrl] + [⇧] はシフトに２つの値を加えたものを設定します。

```
10 KEY 20,CHR$(&H04+&H03)+CHR$(&H0C)
20 ON KEY(20) GOSUB 2000
30 KEY(20) ON
```

K

KEY (n) ステートメント

目的：

キー割り込みを可能，禁止，または保留状態に設定します。

構文：

KEY (n) — ON ——■
KEY (n) — OFF ——■
KEY (n) — STOP ——■

説明：

nには0～25，30，31の範囲の数式で，割り込みを起こすキーの番号を指定します。番号は，それぞれ次のキーと対応します。nに0を指定するとON KEYで設定した割り込みキーすべてに対して，キー割り込みを可能，禁止，保留状態に設定できます。

番号	キー
1～10，30，31	PF1～PF12
11	↑
12	←
13	→
14	↓
15～25	KEYで定義したキー

KEY (n) ONは，指定されたキーの割り込み処理をを可能な状態に設定します。次の2つの条件が揃うと，BASICはステートメントを1つ処理するたびにこのキーが押されたか否かを検査し，押された場合は割り込み処理を実行します。

1 前もって，ON KEY (n) ステートメントにより有効な割り込み処理が指定されていること

2 すでに，KEY (n) ON ステートメントによりキー割り込みが使用可能になっていること

KEY (n)　OFF は，指定されたキーの割り込み処理を使用禁止の状態に設定します。ON KEY (n) により有効な割り込み処理が与えられていても，キーによる割り込みは起こりません。また，割り込みが無視されたことの情報も残りません。

KEY (n)　STOP は，指定されたキーの割り込み処理を保留状態に設定します。これにより，キーを押しても割り込みは起こりません。ただし，割り込みが保留されたという情報が残り，後に KEY (n) ON により割り込み可能になると，即座に保留された割り込みが実行されます。

例：

```
10 ON KEY(11) GOSUB 50
20 KEY(11) ON
30 INPUT "カーソル上方移動キーを押した後改行キー";A$
40 END
50 PRINT "カーソル上方移動キー"
60 RETURN
Ok
RUN
カーソル上方移動キーを押した後改行キー?
カーソル上方移動キー
Ok
```

この例では，[↑] キー (11) の入力に対して割り込み処理を行わせています。

K

KILL コマンド

目的：

ディスク上のファイルを削除します。

構文：

KILL — ファイル指定 ——■

説明：

ファイル指定には，「ドライブ指定子：ファイル名」形式のストリング式で，削除するファイルを指定します（パスの指定も可能です）。ドライブ指定子を省略すると，DOSの省略時ドライブが参照されます。詳細は，☞ 解説編　第3章「ファイル指定とパス」を参照してください。

ファイル名は完全な形で指定します。エクステンション付きのファイル名であれば，エクステンションも含めて指定する必要があります。BASICプログラムのファイル（エクステンション.BAS）のように，保管時に自動的に付加されるエクステンションに注意してください。

KILLは，ディスク上のファイルであれば，どの形式のファイルにも有効です。ただし，現在使用中のファイルは指定できません。

KILLに現在使用中（オープンしている）のファイルを指定すると，File already open（ファイルがすでにオープンされている）エラーが起こります。

例：

1

```
200 KILL "A:DATA1"
```

この例はドライブAのDATA1というファイルを削除します。

2

```
KILL "LEVEL1¥LEVEL2¥PROG.BAS"
```

この例は，パス名も含めてファイルを指定しています。

KLEN 関数

目的：

ストリングの文字数を返します。

構文：

―― KLEN (x$) ――

説明：

x$ には，ストリング式を指定します。

印刷不可能な文字（制御文字など）やブランクも，1 文字に数えられます。また，2 バイト・コード文字（連続コード 256 以上）も，1 バイト・コード文字（連続コード 255 以下）と同様に 1 文字として数えられます。

ストリングのバイト数を求めるためには，LEN 関数を使用します（2 バイト・コード文字は 2 として数えられます）。

例：

次の例は，KLEN と LEN の機能の相違を表しています。2 種類のストリングの文字数（KLEN），バイト数（LEN），および 2 バイト・コード文字の数（LEN − KLEN）を表示します。

```
10 A$="This is a book."
20 B$="これは本です。"
30 PRINT A$,KLEN(A$),LEN(A$),LEN(A$)-KLEN(A$)
40 PRINT B$,KLEN(B$),LEN(B$),LEN(B$)-KLEN(B$)
Ok
RUN
This is a book.          15        15        0
これは本です。           7         14        7
Ok
```

K

KPOS 関数

目的：　ストリングのｎ番目の文字の位置を，バイト数で返します。

構文：　—— KPOS (x$, n) ——

説明：　x$ には，ストリング式を指定します。

n は左から数えた文字数により文字の位置を表すもので，数式で指定します。ストリング全体の文字数より n の値が大きい場合，この関数は 0 を返します。

文字の位置 n と KPOS の値の差は，この文字より左にある 2 バイト・コード文字の数を示します。

例：

```
10 A$="6月はJUNE"
20 PRINT KPOS(A$,3)
RUN
 4
Ok
```

この例では，ストリングの 3 番目の文字 "は" の位置を求めています。"6" は 1 バイト，"月" は 2 バイトの長さをもつため，"は" は 4 バイト目から始まります。

KTN$ 関数

目的：　ストリングの最初の文字のJIS区点コードを，ストリング形式で返します。

構文：　**―― KTN$ (x$) ――**

説明：　x$ には，ストリング式を指定します。

この関数は，ストリングの最初が1バイト・コード文字の場合は3桁，2バイト・コード文字の場合は4桁のJIS区点コードを表すストリングを返します（☞解説編　付録D「コード体系」）。

x$ が空ストリングの場合，Illegal function call（違法関数呼出し）エラーが起こります。

ストリングの最初の文字の連続コードを求める場合は，ASC関数を使用します。

例：　次の例は，漢字「日」のJIS区点コード（3892）を表示しています。ストリングの内容が異なっても，最初の文字が同じであれば，同じストリングが返されます。

```
10 A$="日本一":B$="日付"
20 PRINT KTN$(A$),KTN$(B$)
RUN
3892          3892
Ok
```

K

LEFT$ 関数

目的： ストリングの左から n バイト分の文字を返します。

構文： —— LEFT$ (x$, n) ——

説明： x$ は，ストリング式を指定します。

n は結果として返す長さをバイト数で示したもので，0 ～ 255 の数式で指定します（2 バイト・コード文字が含まれる場合，n は文字数とは異なります）。

n がストリング全体のバイト数より大きいと，ストリング全体が結果として返されます。また，n に 0 を指定すると，空ストリング（長さ 0）が返されます。

n で指定した長さが 2 バイト・コード文字の 1 バイト目と 2 バイト目を分断する場合，返されるストリングの最終バイトには，分断された文字の内部コードの 1 バイト目が入ります。ただし，元のストリングには影響を及ぼしません。

同様にストリングの一部分を返す MID$ 関数，および RIGHT$ 関数の記述も参照してください。

例： 1

```
10 A$="BASIC PROGRAM"
20 PRINT LEFT$(A$,5)
30 PRINT A$
RUN
BASIC
BASIC PROGRAM
Ok
```

2

```
10 DEF FNKLEFT$(X$,N)=LEFT$(X$,(KPOS(X$,N+1)-1))
20 X$="左側から 2 文字"
30 PRINT X$
40 PRINT FNKLEFT$(X$,2)
RUN
左側から 2 文字
左側
Ok
```

この例のユーザー関数 FNKLEFT$ は，ストリングの左から n 文字を返します。

L

LEN 関数

目的：　ストリングの長さ（バイト数）を返します。

構文：　**—— LEN (x$) ——**

説明：　x$ は，ストリング式を指定します。

印刷不可能な文字（制御文字など）やブランクも，バイト数に数えられます。

例：

```
10 A$="6月は JUNE"
20 PRINT LEN(A$)
RUN
 10
Ok
```

“は”と“J”の間のブランクも 1 バイトとして数えられ，全体で 10 バイトになります。

```
10 '--- 小文字を大文字に ---
20 A$ = "abcde"
30 FOR I = 1 TO LEN(A$)
40 C = ASC(MID$(A$,I,1))
50 IF 97 <= C <= 122 THEN C=C-32
60 B$ = B$ + CHR$(C)
70 NEXT I
80 PRINT A$;B$
Ok
RUN
abcdeABCDE
Ok
```

LET ステートメント

目的：

式の値を変数に割り当てます（代入）。

構文：

LET **変数＝式**

（LET は省略可能）

説明：

LET は省略が可能です。

変数には，**式**の値を受け取る変数，または配列要素の名前を指定します。

式には，**変数**に割り当てるストリング式，または数式を指定します。

数値またはストリングに関して**変数**と**式**の型が一致しない場合，Type mismatch（型が不一致）エラーが起こります。

例：

```
110 LET A=12
120 LET B=A+2
130 LET C$="ABC"
```

この例は，変数 A に値 12 を割り当て，次に変数 B に A + 2（つまり 14）を割り当て，最後に変数 C$ にストリング “ABC” を割り当てています。これを，次のように書き表すこともできます。

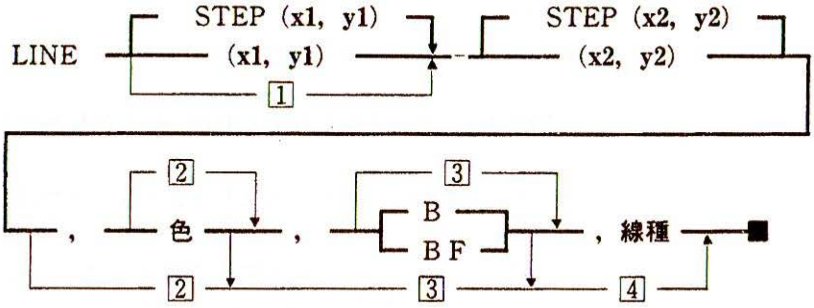
```
110 A=12
120 B=A+2
130 C$="ABC"
```

LINE ステートメント

目的：

各種の線（実線，破線，一点鎖線など）を使って，画面上に直線または矩形を描きます。グラフィック・モードでのみ使用できます。

構文：



1 最終座標
2 前景色
3 線を描く
4 実線

説明：

(x1, y1) または STEP (x1, y1) には線を描き始める位置を指定し，(x2, y2) または STEP (x2, y2) には線を描き終える位置を，指定します。いずれの座標指定も，STEP 付きは相対座標による指定，STEP なしは絶対座標による指定と判断されます（☞解説編 第 3 章の「座標指定」）。

1 つ目の座標を省略すると，これまでの最終座標から線が描かれます。2 つ目の座標とその前の**ハイフン (-)** は省略できません。

色は，線の色のパレット番号を指定します。

画面モード	パレット番号
単色	(0/1)
4 色カラー	0 ～ 3
16 色カラー	0 ～ 15

色の指定を省略すると，現在の前景が使用されます。

B および BF は，2 座標を対角の頂点とする矩形を描く指定です。B は線で矩形を描き，BF は色で中塗りした矩形を描きます。B および BF を省略すると，2 座標を結ぶ直線を描きます。

線は，16 進表示 H0000 ～ HFFFF（10 進表示 32768 ～ 32767）の数式で線の種類を指定します。省略時は実線になります。

線の種類は 16 個のドット（点）の組み合わせで表され，この 16 ドットの繰り返しで線が描かれます。線に指定する数値（16 ビットで内部表現される）は，各ビットの値を 16 個のドットに対応させて線の種類を表現します。ビット値が 1 の桁は色のドットと，ビット値が 0 の桁は無表示のドットと，それぞれ対応します。次の図は，ドットとビットの対応例です。

(実線)

1111111111111111
(&HFFFF)

(荒い破線)

1111100011111000
(&HF8F8)

(細かい破線)

1010101010101010
(&H5555)

線の指定に関して，次の点に注意が必要です。

- 無表示のドットとは，その位置を飛ばすことを示します。つまり，何らかの絵が描かれている箇所に無表示のドットを配置しても，もとの絵のドットが表示されます。

- 中塗り矩形に対して線を指定すると，実行はされますが Syntax error（構文の誤り）となります。

- 16 進数&H0000 ～&HEFFF は 10 進数 0 ～ 32767 と対応し，&H8000 ～&HFFFF は -32768 ～ -1 と対応します（10 進表示では 16 ビットの内 1 ビットが符号を示します）。10 進数で 32768 以上を指定すると，Overflow（桁あふれ）となります。

- 線は，通常上から下へ描かれます。水平線は終点（x_2，y_2）から開始点（x_1，y_1）方向に描かれます。ただし，終点（x_2，y_2）に範囲外の座標を指定した場合は，開始点（x_1，y_1）から終点（x_2，y_2）方向に描きます。したがって，水平線の破線を描く場合で，終点に範囲外の座標を指定した場合は，ビット列が逆転します。

なお，LINE で描かれる図の中で，画面の座標範囲から出る部分は無視されます。

例：

1
```
LINE -(200,200)
```

これは，これまでの最終座標から点（200,200）まで，前景色の実線を引きます。

2
```
LINE (100,100)-(200,200),2,BF
```

これは，点（100,100）と点（200,200）を対角の頂点とする矩形を描き，パレット番号 2 番の色を塗ります。

3
```
LINE (100,100)-STEP(50,-50),,B,&HF99F
```

これは，点（100,100）と点（150,50）を対角の頂点とする矩形を，前景色の 1 点鎖線（-・--・-）で描きます。

4
```
10 SCREEN 1,0
20 FOR I=0 TO 15
30 J=2^I+J
40 IF J>32767 THEN L=J-2^16 ELSE L=J
50 LINE (0,300+I*5)-(500,300+I*5),,,L
60 NEXT I
70 END
Ok
```

このプログラムは，FOR..NEXT ループを使って破線の種類を変えています。線の種類を 10 進数で指定するため，値が 32767 を超えたら 2 の 16 乗を引いて負数に変えている点に注意してください。

LINE INPUT ステートメント

目的：

キーボードから，行単位（半角文字で 255 文字まで）のストリングを変数に読み込みます。

構文：

LINE INPUT [;] ["プロンプト";] ストリング変数

説明：

"**プロンプト**"には，プロンプト（キーボードからの入力を要求するメッセージ）に表示するストリング定数を指定します。INPUT ステートメントと異なり，これを省略しても疑問符は表示されません。

変数には，読み取ったストリングを受け取るストリング変数，または配列要素を指定します。

キーボードから文字を入力すると，[↵] キーが押されるまで，1 行のデータとして LINE INPUT ステートメントに読み取られて**変数**に割り当てられます。ただし，[↵] キーのコード（改行/復帰文字）は**変数**には割り当てられません。

LINE INPUT の直後に**セミコロン（;）**を指定すると，[↵] キーを押しても画面上のカーソルの改行/復帰動作は行われず，ユーザーが応答した行にとどまります。この場合も改行/復帰文字は**変数**に送られず，[↵] キーはキーボード入力行の終わりを示すだけの働きになります。

LINE INPUT へのデータ入力を途中で取り消すためには，[Break] キー機能を押します。これにより BASIC はコマンド入力のレベルに戻り，Ok を表示します。ここで CONT コマンドを指定すると，LINE INPUT への入力からプログラムを再開することができます。

例：

LINE INPUT#ステートメントの例を参照してください。

LINE INPUT#ステートメント

目的：

順次ファイルから，行単位（半角文字で 255 文字まで）のストリングを変数に読み込みます。

構文：

LINE INPUT — #ファイル番号，ストリング変数 ——■

説明：

ファイル番号は，順次ファイルのオープンに使用したファイル番号を指定します。また，順次アクセスするランダム・ファイルも指定することができます。

変数には，読み取ったストリングを受け取るストリング変数，または配列要素を指定します。

LINE INPUT#によりデータを読み込むと，復帰文字が読み込まれるまで，1 行のデータとして**変数**に割り当てられます。1 行のデータが復帰文字で終わった場合，復帰文字自体は**変数**には入りませんが，次の LINE INPUT# は復帰文字の次のデータから読み込みます。

LINE INPUT#は，各項目のデータが行単位に入っているファイルや，文字形式のプログラム自身をデータとして他のプログラムに読み込む場合に役立ちます。

例：

次のプログラムは，LINE INPUT でキーボードから読み込んだデータを順次ファイルに書き込み，次にこのファイルから同じデータを LINE INPUT#で読み込んでいます。通常は区切り文字として扱われるカンマ（,）やブランクも，ストリングの一部として読み込まれている点に注目してください。

```
10 OPEN "LIST" FOR OUTPUT AS#1
20 LINE INPUT "住所は?";C$
30 PRINT #1,C$
40 CLOSE #1
50 OPEN "LIST" FOR INPUT AS#1
60 LINE INPUT #1,C$
70 PRINT C$
80 CLOSE #1
Ok
RUN
住所は?
```

「港区 六本木,」

```
住所は?港区 六本木,
港区 六本木,
Ok
```

LIST コマンド

目的：

現在，記憶域にあるプログラムを，画面上または指定した装置に出力します。

構文：

説明：

プログラム中の出力する部分の最初の行番号を行 **1** に，最後の行番号を行 **2** に，0 〜 65529 の数式で指定します。**ピリオド (.)** は，現在行を示します。

行番号の指定方法（ピリオドによる指定も同様）により，次の範囲が出力されます。

行 1	その行だけ
行 1 −	その行からプログラムの最後まで
− 行 2	プログラムの最初からその行まで
行 1 − 行 2	行番号 1 から行番号 2 まで

ファイル指定は，ストリング式でファイルを指定します☞ 解説編　第　章「ファイル指定とパス」。**ファイル指定**を省略すると，画面上に出力（表示）されます。

画面またはプリンターへの出力は，Break キー機能で中断することができます。

ディスク上のファイルを指定すると，ファイル名にエクステンション.BAS が自動的に付加され，プログラムが文字形式で保管されます。

例：

1 LIST

プログラム全体を画面に表示します。

2 LIST 10-20,"LPT1;"

行 10 から行 20 をプリンターに出力します。

3 LIST 100-,"COM1:1200,E,8"

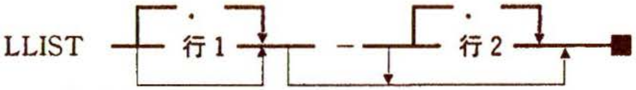
プログラムの行 100 から最後までを，次の条件で通信カードに送ります。

データ転送速度	：1200bps
パリティ	：偶数パリティ
データ・ビット	：8 ビット
ストップ・ビット	：1 ビット

LLIST コマンド

目的： 現在記憶域にあるプログラムを，プリンターに出力します。

構文：



説明： プログラム中の出力する部分の最初の行番号を**行 1** に，最後の行番号を**行 2** に，0 ～ 65529 の数式で指定します。**ピリオド (.)** は，現在行を示します。

行番号の指定方法（ピリオドによる指定も同様）により，次の範囲が出力されます。

行 1	その行だけ
行 1 －	その行からプログラムの最後まで
－ 行 2	プログラムの最初からその行まで
行 1 － 行 2	行番号 1 から行番号 2 まで

LLIST の実行後は，常にコマンド入力のレベルになります。

例：

1

```
LLIST
```

プログラム全体を印刷します。

2

```
LLIST 10-20
```

10 から行 20 を印刷します。

L

LOAD コマンド

目的：

プログラムを，指定した装置から記憶域に読み込みます。また，このプログラムを実行させることもできます。

構文：

LOAD — ファイル指定 [**, R**] ■

説明：

ファイル指定には，「装置名：」または「ドライブ指定子：ファイル名」形式のストリング式で，ファイルを指定します（ファイル名については，☞解説編　第 3 章「ファイル指定とパス」)。

装置名またはドライブ指定子を省略すると，省略時ドライブの現行ディレクトリーが参照されます。パス名も指定できます。

エクステンションなしのファイル名は，エクステンション.BAS 付きで参照されます。

ファイル指定に間違いがある場合，Bad file name（ファイル名の誤り）エラーとなり，読み込みは取り消されます。

R は，読み込んだプログラムを，引き続き実行することを指定します（これは「RUN ファイル指定」と同じ働きです)。

LOAD は，指定されたプログラムを読み込む前に，記憶域内の変数とプログラムを削除します。また，R 指定がない場合は，オープンしているデータ・ファイルもクローズします。R を指定した場合は，ファイルのクローズは行いません。したがって，R 付きの LOAD は，プログラムの連鎖や，セグメント化されたプログラムの実行に役立ちます。

例：

1 LOAD "MENU"

MENUというプログラムを読み込みます。

2 LOAD "在庫管理",R

プログラム「在庫管理」を読み込み，実行します。

3 LOAD "B:報告書.BAS

プログラム「報告書」をドライブBから読み込みます。エクステンション.BASは省略が可能です。

L

LOC 関数

目的:

ファイル中の現在の参照位置を返します。

構文:

—— LOC（**ファイル番号**）——

説明:

ファイル番号には，ファイルのオープン時に使用した番号を指定します。

ランダム・ファイルの場合，LOC は最後に参照（読み込み，または書き込み）されたレコードの**レコード番号を返します。**

順次ファイルの場合，LOC は現在までに読み込んだ，または書き込んだレコードの数を返します（1 レコードは 128 バイトにブロック化されたデータです)。順次入力用のファイルをオープンすると，同時に最初のブロックが読み込まれます。したがって，入力用ステートメントを実行する前でも，LOC は 1 を返します。

通信ファイルの場合，LOC は読み取り待ちの**入力バッファー内のバイト数を返します。**入力バッファーのサイズは 256 バイトですが，BASIC を始動するときに/C: を用いて値を指定すると，バッファーのサイズを変えることができます。

例:

1

```
200 IF LOC(1)>50 THEN STOP
```

この例は，ファイルの 51 番目のレコードを読み込むと，プログラムを停止させます。

2

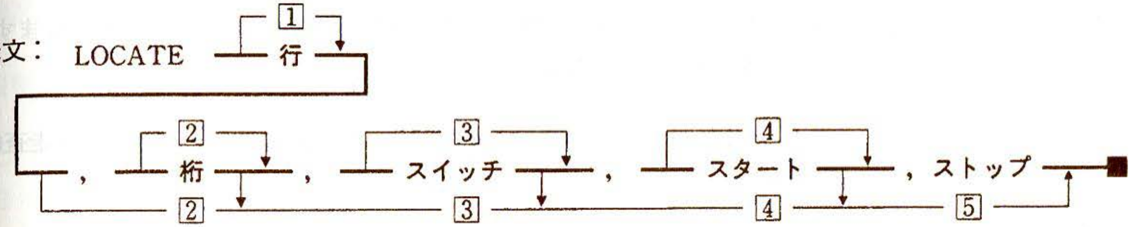
```
300 PUT #1,LOC(1)
```

この例では，読み込んだばかりのレコードを，そのまま書き出しています。

LOCATE ステートメント

目的： 画面上のカーソルの位置を指定します。カーソルの大きさや明滅を指定します。

構文：



1 2 3 4 現在の値

5 **スタート**の値

説明： **行**は，カーソル位置の行番号 1 ～ 24 を指定します。KEY ON のときは 1 ～ 23 です。指定できる値の範囲は WIDTH と VIEW PRINT で指定した行数に依存します。

桁は，カーソル位置の桁番号 1 ～ n を指定します。n は画面により次のように異なります。指定できる値の範囲は WIDTH で指定した桁数に依存します。

文字モード……………………………………… 80
4 色カラー**以外**のグラフィック・モード… 78
4 色カラー・グラフィック・モード……… 40

スイッチは，カーソルを表示するか否かを，次の 0 または 1 で指定します。

0: カーソルは表示されない
1: カーソルが表示される

スタートは，カーソルの四角い部分の始まりの位置を，0 ～ n で指定します。n は，ディスプレイや画面モードにより，次のように異なります。

16ドット・システム
　文字モード………………………… 20
　グラフィック・モード…………… 19
24ドット・システム
　文字/グラフィック・モード …… 28

ストップは，カーソルの四角い部分の終わりの位置を，0～nで指定します。nは**スタート**と同じです。

スタートは，カーソルの開始走査線です。**ストップ**は，カーソルの停止走査線です。

カーソル

カーソルの明滅は，文字モードのときに，**スタート**に次の値を加えます。カーソルを「表示しない」（**スイッチ**が1でも）指定もできます。

加える値	カーソルの状態
64	速く明滅する
96	ゆっくりと明滅する
32	表示されない

スタートを指定して，**ストップ**を省略すると，**ストップ**には，**スタート**の値が設定されます。**スタート**が**ストップ**より大きいと，値が入れ替って設定されます。

スタートに0～127以外，**ストップ**に0～31以外の値を指定したり，他のパラメーターに範囲外の値を指定すると，Illegal function call（違法関数呼び出し）エラーになります。

例：

`LOCATE 1,1`	カーソルをホーム・ポジション（画面の左上隅）に移動します。
`LOCATE,,,0,7`	開始走査線を 0 に，停止走査線を 7 に，指定しカーソルを小さくします。
`LOCATE,,0`	カーソルを「表示しない」指定をします。
`LOCATE,,1,64,31`	カーソルを速く明滅させます。
`LOCATE,,1,96,31`	カーソルをゆっくり明滅させます。

L

LOCK ステートメント

目的： ファイルの全部または一部に対して，アクセス禁止の指定をします。

構文：

説明： n はオープンしたファイルの番号です。値の範囲は，1 ～ 255 です。

番号は，アクセス禁止の範囲をレコード番号で指定します。値の範囲は 1 ～ 4,294,967,295 です。指定は，ランダム・ファイルとしてオープンしているファイルに対してのみ有効です。

LOCK ステートメントや UNLOCK ステートメントで作成したプログラムを実行する前に，共通モジュールが記憶域中に存在している必要があります。このモジュールは，DOS プロンプトの下で SHARE コマンドを実行することによって，記憶域に配置します。

順次ファイルとしてオープンしたファイルに対してこの指定を行うと，どのような範囲を指定しても，無制限にロックされるため注意が必要です。

最初のレコード番号を省略すると 1 が指定されます。最初のレコードを指定しない場合，1 レコードのみアクセス禁止となります。

ファイルをクローズする前または，プログラムを終了する前に，すべてのアクセス禁止指定を解除しておかないと，予期せぬ実行結果となる場合があります。

LOF 関数

目的： ファイルの長さ（バイト数）を返します。

構文： —— **LOF（ファイル番号）** ——

説明： **ファイル番号**には，ファイルのオープンに，使用した番号を指定します。

LOF は，ファイルに割り振られた実際のバイト数を返します。

通信ファイルの場合は，通信バッファー内の未使用空間のバイト数を返します。通信バッファー・サイズの省略時は，256 バイトです。BASIC 始動時に，選択肢/C: で通信バッファー・サイズを変更できます。

例：

```
10 OPEN "BIG" AS #1
20 GET #1,LOF(1)/128
```

BIG という名前のファイルの最後のレコードを取り出します。BIG は，128 バイトのレコード長で作成されたものです。

L

LOG 関数

目的：　数式の自然対数（底が e の対数）を返します。

構文：　── LOG (x) ──

説明：　x は，1 以上の数式を指定します。

例：

```
PRINT LOG(45/7)
860752
Ok
```

457 を対数計算したものです。

```
E=2.718282
 1
Ok
PRINT LOG(E*E)
 2
Ok
```

e の対数と e の 2 乗の対数を計算した例です。

LPOS 関数

目的: プリンター・バッファー内のプリント・ヘッドの現在位置を返します。

構文: —— LPOS (n) ——

説明: nは，0または1を指定します。これは，テストするプリンターがLPT1（第1プリンター）であるという指定です。0と1の違いは，ありません。

LPOS関数が返す値は，プリンター・バッファー内の値なので，プリント・ヘッドの物理的位置を与えるとは限りません。

例: この例は，行の長さが60文字を越えると，プリンターに復帰文字を送り，次の行へ，スキップします。

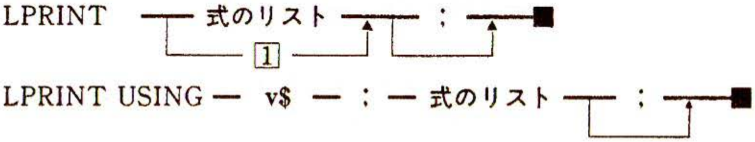
```
100 IF LPOS(0)>60 THEN LPRINT CHR$(13);
```

L

LPRINT およびLPRINT USING ステートメント

目的：　データをプリンターに出力します。

構文：



[1] 復帰/改行を行う

説明：

式のリストには，印刷する数式かストリング式のリストを指定します。式と式の間は，必ずカンマかセミコロンで区切ります。

v$ は印刷書式を，ストリング定数または，ストリング変数で指定します。印刷書式の指定方法は，☞「PRINT ステートメント」と「PRINT USING ステートメント」。

このステートメントは，データをプリンターに出力する以外「PRINT ステートメント」や「PRINT USING ステートメント」と機能が同じです。

LPRINT は，出力幅が半角で 80 のプリンターを想定しています。したがって，半角で 80 文字印刷すると自動的に復帰/改行文字が挿入されます。一行の最後のステートメントに，セミコロンを付けないと 2 行スキップされます。WIDTH"LPT1:" ステートメントを使うと，出力幅を変更できます。

LPRINT と CHR$ を使うと，プリンター制御文字を送ることができます。プリンター制御文字は，☞ 各プリンターの操作ガイドを参照してください。プリンター制御文字とその機能はプリンターにより異なるため注意してください。

例：　次の例は，プリンター制御文字を指定したプログラムとそのプリンター出力です。

```
10 A$="123456789" : B$="あいうえお"
20 LPRINT A$;B$                     '--- 通常の印刷
30 LPRINT A$;STRING$(9,CHR$(8));B$ '--- 後退文字の指定
40 LPRINT A$;CHR$(10);B$            '--- 改行文字の指定
50 LPRINT A$;CHR$(13);B$            '--- 復帰文字の指定
60 LPRINT A$;CHR$(27)+"[";B$;CHR$(27)+"]" '-- 倍角印刷
```

```
123456789あいうえお
123456789お
123456789
         あいうえお
123456789
あいうえお
123456789あ い う え お
```

L

LSET および RSET ステートメント

目的：

ランダム・ファイル・バッファー内にデータを移行します。これは，PUT ステートメントで，ランダム・ファイルに，データを書き込むための準備です。

構文：

LSET — ストリング変数＝x$ —■
RSET — ストリング変数＝x$ ——■

説明：

ストリング変数は，FIELD ステートメントで定義した変数名を指定します。

x$ には，移行するデータをストリング式で指定します。

移行したデータは，ストリング変数で識別された，フィールドに設定されます。LSET は左寄せ，RSET は右寄せで設定されます。余った桁は，ブランクが置かれます。x$ が，ストリング変数より長い場合は，LSET も RSET も x$ の右端から文字が失われます。

数値は，LSET や RSET を行なう前に，ストリングに変換する必要があります。数値からストリングへの変換は，MKI$,MKS$,MKD$ 関数を使ってください。ランダム・ファイルの使用方法の詳細は，☞ 解説編　付録 A「ディスク入出力」。

ランダム・ファイル出力と無関係に，LSET と RSET を，文字の左寄せや右寄せのために使うこともできます。この機能は，印刷書式を作成するときに，便利です。次の例は，ストリング N$ を 20 文字フィールド内で右寄せにします。

```
110 A$=SPACE$(20)
120 RSET A$=N$
```

例：

数値 AMT をストリングに変換し，フィールド A$ 内で左寄せにします。

```
150 LSET A$=MKS$(AMT)
```

MERGE コマンド

目的： 文字形式で保管されたプログラムを，記憶域内のプログラムと重ね合わせます。

構文： **MERGE ― ファイル指定 ―■**

説明： **ファイル指定**は，「装置名：」または「ドライブ指定子：ファイル名」の形式のストリング式で，マージするファイルを指定します（ファイル名については，☞解説編　第3章「ファイル指定とパス」）。

ファイル指定に間違いがあると，Bad file name（ファイル名の誤り）エラーになり，このコマンドは無効になります。

装置名またはドライブ指定子を省略すると，DOS の省略時ドライブの現行ディレクトリーが参照されます。また，パス名を指定することもできます。

マージとは，文字形式で保管しているプログラムを，記憶域内のプログラムと重ね合せることをいいます。同じ行番号があると，文字形式で保管しているプログラムの方が優先されます。したがって，マージするときは，双方のプログラムの行番号に注意してください。

MERGE コマンド実行後，BASIC は，コマンド・レベルに戻ります。

マージするプログラムが，文字形式で保管されたものでない場合，Bad file mode（ファイル・モードの誤り）エラーになります。記憶域内のプログラムは変更されません。

例：

```
MERGE "A:NUMBERS"
```

ドライブ A にある NUMBERS という名前のファイルを，記憶域にあるプログラムにマージします。

MID$ ステートメント

目的：

ストリングを置き換えます。

構文：

MID$ ── (x$, n ─┬─, m ─┬─) = y$ ──■
　　　　　　　　 └─ [1] ─┘

[1] y$の長さ

説明：

ストリング x$ の n バイト目からを，ストリング y$ の先頭から，長さ m バイト分で置き換えます。

n は，1～ストリング x$ の長さまでの範囲の整数式です。

m は，0～255 の整数式です。

m を省略すると，y$ 全部で，x$ を置き換えます。y$ が x$ より長いときは，x$ の長さ分だけ，置き換わります。

注） ストリングに 2 バイト・コードの文字が含まれるときは，文字の境界で置き換えられるように，n と m の指定を行う必要があります。

n または m に，範囲外の値を指定すると Illegal function call（違法関数呼び出し）エラーになります。

例：

```
10 A$="MARATHON, GREECE"
20 MID$(A$,11)="FLORIDA"
30 PRINT A$
```

ストリング A$ の中の文字を置き変えます。置き換えられるストリング A$ の全体の長さは，変わりません。

```
RUN
MARATHON, FLORID
Ok
```

```
10 X$="神奈川県 川崎市"
20 Y$="横浜市"
30 MID$(X$,11,4)=Y$
40 PRINT X$
RUN
神奈川県 横浜市
Ok
```

2バイト・コード文字を含んだストリングの置き換えです。

M

MID$ 関数

目的：

ストリングの中の指定した部分を返します。

構文：

```
── MID$ ─ (x $, n ─┬─ , m ─┬─ ) ──
                   └── [1] ─┘
```

[1] nバイト目以降すべて

説明：

ストリング x$ の n バイト目から，m バイトの長さのストリングを返します。

n は，1 ～ 255 の範囲です。

m は，0 ～ 255 の範囲です。

m を省略したときと，n バイト目以降が，m バイトより少ない場合は，n バイト目以降がすべて返ります。

m が 0 のときと，n が x$ の長さより大きい場合，空ストリングが返ります。

1 バイト・コード文字は，バイト数と文字数が一致しています。2 バイト・コード文字の 1 文字は 2 バイトです。n と m で指定する，ストリングの境界が 2 バイト・コードの途中になる場合は，文字が正しく設定されませんので注意してください。

例：

ストリング B$ の一部分を取り出します。

```
10 A$="GOOD"
20 B$="MORNING EVENING AFTERNOON"
30 PRINT A$;MID$(B$,8,8)
RUN
GOOD EVENING
Ok
```

これは，x$ の n 文字目から m 文字分のストリングを返すために，MID$ 関数を使って，関数 FNKMID$ を作成した例です。

```
10 DEF FNKLEFT$(X$,N)
=LEFT$(X$,-(KLEN(X$)<=N)*LEN(X$)
      -(KLEN(X$)>N)*(KPOS(X$,N+1)-1))
20 DEF FNKMID$(X$,N,M)
      =FNKLEFT$(MID$(X$,KPOS(X$,N)
      -(KPOS(X$,N)=0)*255),M)
30 X$="7文字目から3文字取り出す"
40 PRINT X$
50 PRINT FNKMID$(X$,7,3)
RUN
7文字目から3文字取り出す
3文字
Ok
```

M

MKDIR コマンド

目的：

ディスクに新規のディレクトリーを作成します。

構文：

MKDIR ― パス名 ――■

説明：

パス名には，63 文字以内のストリング式を指定します。現行ドライブ以外のドライブに指定する場合は，「ドライブ指定子：パス名」の形式で指定します。（詳細は，☞ 解説編　第 3 章「ファイル指定とパス」）

このコマンドは，DOS の MKDIR コマンドと同じ機能を持っています。

例：

```
MKDIR "販売"
```

ルート・ディレクトリーに経理というディレクトリーを作成します。

```
MKDIR "販売¥鈴木"
```

販売の下に鈴木というディレクトリーを作成します。

次の 2 通りの方法は，同じディレクトリーを作成します。

1

```
MKDIR "経理"
```

ルート・ディレクトリーに販売というディレクトリーを作成します。

```
CHDIR "経理"
MKDIR "山田"
```

現行ディレクトリーを経理にします。
経理の下に山田というディレクトリーを作成します。

2

```
MKDIR "経理¥山田"
```

ルート・ディレクトリーで，経理の下に山田というディレクトリーを作成します。

MKI$、MKS$、MKD$ 関数

目的：　数値型からストリング型に変換します。

構文：

―― MKI$（整数式）――
―― MKS$（単精度数）――
―― MKD$（倍精度数式）――

説明：　MKI$ は，整数を 2 バイト・ストリングに変換します。
MKS$ は，単精度式を 4 バイト・ストリングに変換します。
MKD$ は，倍精度式を 8 バイト・ストリングに変換します。

これらの関数は，数値をランダム・バッファーに移行するときに使用するとファイルの数値データ領域を節約できます。数値を LSET や RSET ステートメントで移行するには，ストリングに変換しておく必要があるためです。これらの関数は，数値を内部形式のままストリングの領域に複写します。したがって，データのバイト数は変わりません。ストリング領域に複写された数値を数値型に戻すには，CVI，CVS，CVD 関数を使用します。

例：　これは，数値をランダム・バッファーに移行するときの例です。

```
100 FIELD #1,4 AS D$,20 AS N$
110 LSET D$=MKS$(AMT)
120 LSET N$=A$
130 PUT #1
```

行 100……ランダム・ファイルのフィールドを定義する。
行 110……数値を，ストリングに変換し，ランダム・ファイル・バッファーに置く。
行 120……ストリングは，そのままランダム・ファイル・バッファーに置く。
行 130……データを，ランダム・ファイル・バッファーから，ランダム・ファイルに書き込む。

M

NAME コマンド

目的：　ディスク・ファイルの名前を変更します。

構文：　NAME — **旧ファイル指定** — AS — **新ファイル名指定** —■

説明：　**旧ファイル指定**は，名前を変更するファイルを指定します。

新ファイル指定は，新しいファイルの名前を指定します。

旧ファイル指定，新ファイル指定とも，「装置名：」または「ドライブ指定子：ファイル名」の形式のストリング式で指定します。

装置名またはドライブ指定子を省略すると，DOS の省略時ドライブの現行ディレクトリーが参照されます。パス名も指定できます。

新ファイル指定に**旧ファイル指定**と異なるドライブを指定することは，できません。

新ファイル指定にファイルを指定するときは，ファイル名に，省略時のエクステンション.BASが，付けられないので，注意してください。

例：　この例は，ドライブ A にある DOG.BAS というファイルの名前を CAT.BAS という名前にします。

```
NAME "A:DOG.BAS" AS "A:CAT.BAS"
```

この例はドライブ A のディレクトリー JAPAN にある TOKYO というファイルをドライブ A のディレクトリー USA の HAWAII というファイルにします。（ディレクトリー USA は，すでに存在している必要があります。）

```
NAME "A:¥JAPAN¥TOKYO" AS "A:¥USA¥HAWAII"
```

NEW コマンド

目的：　記憶域にあるプログラムを削除し，すべての変数を消去します。

構文：　NEW ──■

説明：　NEWは，通常，新しいプログラムを作成する前に，記憶域を解放するために使用します。

NEWの実行後，BASICは，コマンド・レベルに戻ります。

NEWは，すべてのファイルをクローズします。

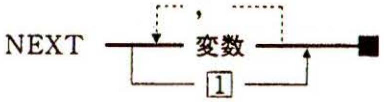
トレースがオン（TRON）のときは，オフ（TROFF）にします。

N

NEXT ステートメント

目的：　FOR・・・NEXT ループを閉じる。

構文：



[1] 最も新しい FOR ステートメントに対応する

説明：　**変数**は，ループ変数を整数変数か単精度変数で指定します。

変数の省略時は，一番近い FOR ステートメントと対応します。

詳細は，☞「FOR および NEXT ステートメント」。

OCT$ 関数

目的：　10 進の数値を 8 進数で表すストリングを返します。

構文：　── OCT$ (n) ──

説明：　n には，-32768 ～ 65535 の 10 進数の数式を指定します。

n が負数の場合，2 の補数が使用されます。したがって，OCT$ (-n) は，OCT$ (65536-n) と同一です。

10 進数から 16 進表示への変換には，HEX$ 関数を使用します。

例：

```
PRINT OCT$(24)
30
Ok
```

10 進数 24 の 8 進表示は 30 です。

O

ON COM (n) ステートメント

目的：

通信バッファーに情報が入った時点で処理する通信割り込みルーチンを指定します。

構文：

ON COM (n) — GOSUB — 行 ——■

説明：

nには，通信カードの番号を指定します。

行には，通信割り込みルーチンが始まる行番号を指定します。行に0を指定すると，通信バッファーに情報が入っても通信割り込みは起こりません。
通信割り込み処理を行うためには，次の条件が必要です。

1. ON COM (n) ステートメントで，割り込みルーチンの開始位置が指定されている（行番号0以外）
2. COM (n) ON ステートメントで通信割り込みが可能な状態に設定されている

条件が揃った時点から，BASICは1ステートメント処理するごとに通信バッファーの中を確認し，情報が入ってきていれば割り込み処理に移ります（割り込みの発生はプログラムの実行中に限られます）。

COM (n) OFF（割り込み無効），またはCOM (n) STOP（割り込み停止）が指定されている間は，割り込みは起こりません。ただしCOM (n) STOPは割り込みの情報を保留するため，この情報に従って，COM (n) ON実行時に即座に割り込みが起こります。

COM (n) STOP中に通信バッファーに入ったデータをプログラムで読み出した結果，COM (n) ONの実行時にバッファー内のデータが無くなっていても，割り込みは起こります。データ無しで割り込みルーチンが処理されることを防ぐため，ルーチンの最初にLOC (n) でデータの有無を確認し，空の場合はルーチンの出口に飛ぶように設計することをお勧めします。

一般に，通信割り込みルーチンは，処理を終わる前にバッファーのすべての

データをプログラムの作業域に読み込むよう設計します。データを1文字ずつ読み込むように設計した場合，データ伝送速度が速いとバッファーがあふれることがあります。

通信割り込みルーチンの実行中はCOM（n）STOPに，ルーチンの処理を終わった時点でCOM（n）ONに，自動的に設定されます（ただし，ルーチンにCOM（n）OFFが含まれている場合はCOM（n）ONには戻りません)。したがって，通信割り込みルーチンの実行中に，さらに通信割り込みが起こることはありません。また，エラー割り込みの実行中は，その他のすべての割り込みが禁止状態になるため，通信割り込みは起こりません。

割り込みルーチンから通常のプログラムの流れに戻るためには，RETURNを使用します。RETURNの後に行番号を指定すると，割り込みが発生した箇所以外に戻ることも可能です。ただし，割り込みが起こった時点で実行されていたGOSUB，WHILE，FORなどはいずれも有効で，割り込みから戻るとこれが続行されるため，割り込み箇所以外に戻る場合は注意が必要です。

例：

```
150 ON COM(1) GOSUB 500
160 COM(1) ON
.
.
500 REM 通信割り込みルーチン
.
.
590 RETURN
```

この例では，行500～590が通信割り込みルーチンです。

O

ON ERROR ステートメント

目的： エラーが発生した場合に処理するエラー割り込みルーチンを指定します。

構文： ON ERROR GOTO ─ 行 ──■

説明： 行には，エラー割り込みルーチンが始まる行番号を指定します。存在しない番号を指定すると，Undefined line number（行番号が不定）エラーが起こります。行に 0 を指定すると，エラー割り込みは起こらずに，エラー・メッセージを表示した後，実行を停止します。

0 以外の行番号でエラー割り込みルーチンが指定された場合，直接モード間接モードにかかわりなく，以後エラーが発生すると必ずエラー割り込みルーチンが実行されます。

ON ERROR で指定したルーチンはすべてのエラーに対して割り込みを起こしますが，エラーの種類が回復不可能なものである場合，ここで実行を停止させる必要があります。そのため，割り込みルーチン内でエラーの種類を判別し，回復不可能な場合は ON ERROR GOTO 0 で実行を停止するよう設計されることをお勧めします。

エラーが回復可能なもので，回復後プログラムの流れに戻る場合は，RESUME を指定します（☞ RESUME ステートメント）。

エラー割り込みルーチン内でエラーが起こった場合，割り込みは発生しません。したがって，エラー・メッセージを表示した後，プログラムの実行は停止します。

例： ☞ ERR，ERL 変数の例。

ON...GOSUB, ON...GOTO ステートメント

目的： 式の値に従って，複数の行番号の内の 1 つに分岐します。

構文：

```
ON ― n ― GOSUB ―┬― 行 ―┬――■
                 └―― , ←┘

ON ― n ― GOTO ―┬― 行 ―┬――■
                └―― , ←┘
```

説明： n には，0 ～ 255 の範囲の数式を指定します。この値が整数でない場合，四捨五入された結果が用いられます。範囲外の値を指定すると，Illegal function call（違法関数呼出し）エラーが起こります。

行には，分岐先の行番号をカンマ（,）で区切って必要な数だけ指定します。n の値と行の順番はそれぞれ対応し，n の値に従って n 番目の行番号に分岐します。また，n の値が行と対応しない（0 や，行番号の個数より大きい）場合，分岐せずに次の処理が続けられます。

分岐先がサブルーチンであるか一般のステートメントであるかにより，GOSUB または GOTO を指定します。ON...GOSUB から次のステートメントに戻るためには，GOSUB ステートメントの場合と同様に，サブルーチン内に RETURN が必要です。

例：

```
100 ON  L+1 GOTO 150,300,320,390
```

この例では，L の値に従って次の行に分岐します。

L の値	0	1	2	3	4	5 以上
分岐先の行	次の行	150	300	320	390	次の行

ON KEY (n) ステートメント

目的：

指定したキーが押された場合に処理するキー割り込みルーチンを指定します。

構文：

ON KEY（n） — GOSUB — 行 —■

説明：

n は 1 ～ 25 の範囲の数式で，割り込みを起こすキーを指定します。n の値は，次のキーと対応しています。

n	キー
1 ～ 10，30，31	PF1 ～ PF12
11	↑
12	←
13	→
14	↓
15 ～ 25	KEY で指定したキー

行には，キー割り込みルーチンが始まる行番号を指定します。行に 0 を指定すると，キーを押しても割り込みは起こりません。

キー割り込み処理を行うためには，次の条件が必要です。

1. ON KEY（n）ステートメントで，割り込みルーチンの開始位置が指定されている（行番号 0 以外）
2. KEY（n）ON ステートメントでキー割り込みが可能な状態に設定されている

条件が揃った時点から，BASIC は 1 ステートメント処理するごとに指定したキーが押されたか否かを確認し，キーが押された場合は割り込み処理を行います（キー割り込みの発生は，プログラム実行中に限られます）。

KEY (n) OFF (割り込み無効), または KEY (n) STOP (割り込み停止) が指定されている間は, 割り込みは起こりません。ただし KEY (n) STOP は割り込みの情報を保留するため, この情報に従って, KEY (n) ON 実行時に即座に割り込みが起こります。

キー割り込みルーチンの実行中は KEY (n) STOP に, ルーチンの処理を終わった時点で KEY (n) ON に, 自動的に設定されます (ただし, ルーチンに KEY (n) OFF が含まれている場合は KEY (n) ON には戻りません)。したがって, キー割り込みルーチンの実行中に, さらに同じキーで割り込みが起こることはありません (異なるキーの割り込みは可能です)。また, エラー割り込みの実行中は, その他のすべての割り込みが禁止状態になるため, キー割り込みは起こりません。

割り込みルーチンから通常のプログラムの流れに戻るためには, RETURN を使用します。RETURN の後に行番号を指定すると, 割り込み発生が発生した行以外に戻ることも可能です。ただし, 割り込みが起こった時点で実行されていた GOSUB, WHILE, FOR などはいずれも有効で, 割り込みから戻るとこれが続行されるため, 割り込み発生箇所以外に戻る場合は注意が必要です。

例：

```
100 SCREEN 0,0
110 PRINT "PF1 : 旋律 1 PF2 : 旋律 2 PF3 : 終了"
120 ON KEY(1) GOSUB 190      '---PF1 :旋律 1
130 ON KEY(2) GOSUB 210      '---PF2 :旋律 2
140 ON KEY(3) GOSUB 230      '---PF3 :終了
150 KEY(1) ON
160 KEY(2) ON
170 KEY(3) ON
180 GOTO 120
190 PLAY"G.C.A.F.E.C.D..."
200 RETURN
210 PLAY"CEE.EGG.DFF.ABB."
220 RETURN
230 END
```

O

ON PLAY ステートメント

目的：

プログラム実行の背景で音楽演奏を継続するための，音楽割り込みルーチンを指定します。

構文：

ON PLAY（n）— GOSUB — 行 ——■

説明：

ON PLAY（n）は，背景音楽バッファー内の音符の数がn以下になると割り込み処理を行います。nには，1以上の音符数を指定します。バッファー内に置ける音符の最大数は，1つの音符を表す情報量に依存します。nにバッファー内にある音符の数以上の値を指定してもエラーにはなりませんが，割り込みを起こすことはできなくなります。nに1以下の値を指定すると，Illegal function call（違法関数呼び出し）エラーになります。

行には，割り込みルーチンが始まる行番号を指定します。行に0を指定すると，音楽割り込みは起こりません。

音楽割り込み処理を行うためには，次の条件が必要です。

1. ON PLAY（n）ステートメントで，割り込みルーチンの開始位置が指定されている（行番号0以外）。
2. PLAY ON ステートメントで割り込みが可能な状態に設定されている。

条件が揃った時点から，BASICは1ステートメント処理するごとに背景音楽バッファーの内容を確認し，音符の数がn個以下に減っていると割り込み処理を行います（割り込みの発生はプログラム実行中に限られます）。

PLAY OFF（割り込み無効），またはPLAY STOP（割り込み停止）が指定されている場合，割り込みは起こりません。ただしPLAY STOPは割り込みの情報を保留するため，この情報に従って，PLAY ON実行時に即座に割り込みが起こります。

音楽割り込みルーチンの実行中は，PLAY STOPに設定され，割り込み処理を終わった時点では，PLAY ONに設定されます（ただし，ルーチンにPLAY OFFが含まれている場合はPLAY ONにはなりません）。したがって，音楽割り込みルーチンの実行中に，音楽割り込みが起こることはありません。また，エラー割り込みの実行中は，その他のすべての割り込みが禁止状態になるため，音楽割り込みは起こりません。

割り込みルーチンから通常のプログラムの流れに戻るためには，RETURNを使用します。RETURNの後に行番号を指定すると，割り込み発生が発生した行以外に戻ることも可能です。ただし，割り込みが起こった時点で実行されていたGOSUB，WHILE，FORなどはいずれも有効で，割り込みから戻るとこれが続行されるため，割り込み発生箇所以外に戻る場合は注意が必要です。

この他に，音楽割り込みを使用する際は次の点に注意してください。

- 音楽割り込みは，背景音楽モード（PLAY"MB..."）の場合のみ有効です。

- PLAY ONを実行する時点で，すでにバッファー内の音符がn個より少なかった場合，割り込みは起こりません。

- nの値は少なめに指定することをお勧めします。nに大きな値を指定すると，バッファーの中の音が，n以下である確率が多くなるため，割り込みが頻繁に発生します。この結果，割り込み処理以外の実行時間が少なくなり，前景処理が進まなくなります。

O

例：　　次のプログラムはプログラム実行中に背景音楽を継続させている例です。

```
100 COUNT = 0
110 DATA L803CL803CL803FL803GL803AL803EL803FL803A.L803GL803G..
120 DATA L803CL803CL803GL803AL803A#L803GL803AL803A#.L803AL803A..
130 DATA L803AL803G#.L1603AL804C.L803A#L803GL803F#.L1603GL803A#.L803A
140 DATA L803FL803E.L1603FL803A.L803GL803F...
150 GOSUB 160 : GOTO 190
160 IF COUNT = 4 THEN RESTORE 110 : COUNT = 0
170 READ ONPU1$  : COUNT = COUNT + 1
180 PLAY "MB XONPU1$;" : RETURN
190 ON PLAY(1) GOSUB 160
200 PLAY ON
210 CLS
220 LOCATE 10,14
230 PRINT "プログラム実行中に繰返し背景音楽を演奏します。";
240 GOTO 240
```

ON TIMER (n) ステートメント

目的:

一定の時間が経過した時点で処理する割り込みルーチンを指定します。

構文:

ON TIMER (n) — GOSUB — 行 ——■

説明:

nには，経過時間を秒で表す数式を，1～86400の範囲で指定します（86400秒=1日)。nに範囲外の数値を指定すると，Illegal function call（違法関数呼出し）が起こります。

行には，割り込みルーチンが始まる行番号を指定します。行に0を指定すると，割り込みは起こりません。割り込み処理を行うためには，次の条件が必要です。

1. ON TIMER ステートメントで，割り込みルーチンの開始位置が指定されている（行番号0以外）
2. TIMER ON ステートメントで割り込みが可能な状態に設定されている

条件が揃った時点から BASIC は経過時間を計り始め，n秒経過したときに割り込み処理を行います（割り込みの発生はプログラム実行中に限られます)。また，割り込みが起こった時点で，経過時間は再び0から計り直されます。したがって，TIMER ON が指定されると，n秒ごとに割り込みルーチンが実行されることになります。

TIMER OFF（割り込み無効），または TIMER STOP（割り込み停止）が指定されている場合，割り込みは起こりません。ただし TIMER STOP は割り込みの情報を保留するため，この情報に従って，TIMER ON 実行時に即座に割り込みが起こります。

割り込みルーチン実行中は TIMER STOP に，割り込み処理を終わった時点で TIMER ON に，自動的に設定されます（ただし，ルーチンに TIMER OFF が含まれている場合は TIMER ON には戻りません)。したがって，時間の割り込みルーチン実行中は，時間による割り込みは起こりません。また，エラー割り込みの実行中は，その他のすべての割り込みが禁止

O

状態になるため，時間の割り込みは起こりません。

割り込みルーチンから通常のプログラムの流れに戻るためには，RETURNを使用します。RETURN の後に行番号を指定すると，割り込み発生が発生した行以外に戻ることも可能です。ただし，割り込みが起こった時点で実行されていた GOSUB，WHILE，FOR などはいずれも有効で，割り込みから戻るとこれが続行されるため，割り込み発生箇所以外に戻る場合は注意が必要です。

例：

次の例では，画面の左上隅に時刻を表示するために，1 分ごとに割り込みを起こします。

```
10 ON TIMER(1) GOSUB 10000
20 TIMER ON
30 FOR I=1 TO 10000:NEXT
40 END
10000 OLDLIN=CSRLIN
10010 OLDCOL=POS(0)
10020 LOCATE 1,1:PRINT TIME$;
10030 LOCATE OLDLIN,OLDCOL
10040 RETURN
```

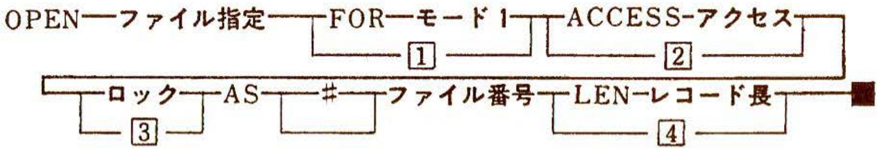
注) 行 10000 と 10010 で現在のカーソル位置を記憶して，行 10020 で時刻を表示した後，行 10030 でカーソルを元に戻しています。

OPEN ステートメント

目的： ファイルや装置への入出力ができるように初期設定します。

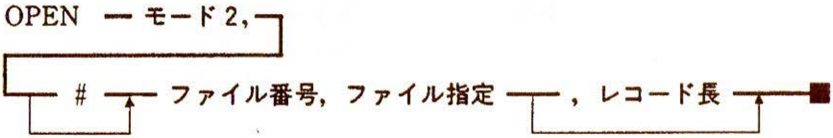
構文：

1



[1] RANDOM
[2] アクセスの説明を参照
[3] LOCK READ WRITE
[4] 128

2



説明：

ファイル指定には，「装置名：」または「ドライブ指定子：ファイル名」形式のストリング式でファイルを指定します（ファイル名については，☞ 解説編 第3章「ファイル指定とパス」）。装置名またはドライブ指定子を省略すると，省略時ドライブの現行ディレクトリーが参照されます。パス名を指定することもできます。

モード 1 はプログラム中のファイルのアクセス方法を，下記のいずれかのモード名を使用して指定します（引用符なしで指定します）。**モード 2** は，ファイルのアクセス方法を，任意のストリング式の最初の 1 文字で指定します（この指定には引用符が必要です）。

アクセス方式	モード 1	モード 2
ランダム入出力：レコード番号による入出力	RANDOM	R
順次出力：ファイルの先頭から順に出力	OUTPUT	O
順次入力：ファイルの先頭から順に入力	INPUT	I
順次出力（付加）：既存レコードの次から順に出力	APPEND	A

次の**アクセス**と**ロック**を使用する場合は，ファイルを共有化するためにDOSのSHAREコマンドを事前に読みこんでおく必要があります。

アクセスは現在使用されている可能性があるファイルをオープンするときにどのような状態でオープンするかを指定します。対象となるファイルが書き込み拒否のモードでオープンされている場合，ファイルのオープン時にPermission denied（書き込み要求拒否）のようなエラーになります。**アクセス**は次のいずれかを指定します。

READ　　現在オープンされている可能性のあるファイルに対して読み取りモードでオープンすることを要求します。読み取りのみ可能となります。

WRITE　　現在オープンされている可能性のあるファイルに対して書き込みモードでオープンすることを要求します。書き込みのみ可能となります。

READ WRITE　　現在オープンされている可能性のあるファイルに対して読み取り/書き込み可能でオープンすることを要求します。読み書きの両方が可能となります。

アクセスを指定せずにアペンド・モードでオープンした場合，BASICはREAD WRITEでオープンしようと試みます。これは，アペンド・モードではデータを書き込む前にファイルの終り（EOF）がどこか読み取る必要があるためです。

アクセスを指定せずにランダム・ファイルをオープンしようとした場合，BASICはまずREAD WRITEでオープンしようと試みます。これが可能でな

いときは，WRITEで試み，最後はREADで試みます。したがってREAD WRITEでオープンできた場合は，GET，PUT が可能ですが，READでオープンされた場合は GET のみが可能となり，WRITEでオープンされた場合は，PUT のみが可能となります。

ロックには，これからオープンしようとしているファイルに対して，他の処理からファイル・オープンの要求が行なわれた場合に，どのような対処を行うかを指定します。指定は次のいずれかです。

SHARED　どのようなプログラムもこのファイルに対して読み書きできます。

LOCK READ　どのようなプログラムもこのファイルに対して読み取りの要求はできません。

LOCK WRITE　どのようなプログラムもこのファイルに対して書き込みの要求はできません。

LOCK READ WRITE
どのようなプログラムもこのファイルに対して読み書きできません。

ロックを指定しないでファイルをオープンした場合，このファイルがオープンしている間はずっと，このファイルに対して他の処理がファイル・オープンすることはできません。すなわち，LOCK READ WRITEに設定されます。

複数の処理プログラムが同じファイルに対して同時にアクセスする可能性がある場合には，上記のようなファイルを防御するためのパラメーターを用いて，処理中に仮定していたデータの内容が，他の処理によって変更されないような措置をとっておく必要があります。ただし，データの内容が刻一刻と書き替えられるべき形のもので，その最新情報を活用する場合などはこの限りではありません。どのような場合であっても，必要なデータを誤って失わないようにアクセスの制御を行うことが必要です。

ファイル番号には 1 ～ 255 の範囲の整数式を指定します。**ファイル番号**は，オープンされてからクローズされるまでの間，そのファイルを参照するために用いられる番号です。

レコード長は，ランダム・ファイルのレコード長を示すもので 1 ～ 32767 の範囲の値を指定します（ただし，BASIC コマンドに/S: を指定した場合，この値を越えない範囲で指定します）。順次ファイルのレコードは可変長であるため，**レコード長**の指定は無効です。ランダム・ファイルでレコード長を省略すると，128 バイトとして扱われます。

OPEN を実行すると，ファイルまたは装置の入出力バッファーが割り振られ，アクセス方式とファイル参照用の番号が決められます。ファイル番号を使用した入出力ステートメントや関数を実行する場合は，必ず事前に OPEN が実行されている必要があります。

同時にオープンできるファイルの最大数は DOS の CONFIG.SYS ファイルの FILES= に指定されている値に依存します。ただし，FILES= に指定されている値の数分だけ BASIC のプログラムでオープンできるのではなく，いくつかは標準入出力など使用されます。このすでに使用されているファイルの数は，システムの状況により異なります。通常，FILES= には BASIC のプログラムでオープンするファイルの数より，3 ～ 5 多い数を指定してください。BASIC では，FILES= に充分な数を指定した場合，最大 15 まで同時にオープンできます。

モードによるアクセス方式の指定は次の点に注意してください。

- ランダム・アクセスは，ディスク・ファイルまたはプリンターに限られます。プリンターを順次出力で使用すると，レコードの最後を示す復帰文字（連続コード 13）の後に，強制的に改行文字（連続コード 10）が付加されます。これに対して，プリンターを幅 255 のランダム・ファイルで使用すると，改行文字は付加されません。このため，プリンターのランダム・アクセスは，同一行に重ねて印刷する場合などに使用します。

- 付加順次出力（APPEND）は，ディスク・ファイルに限られます。次のレコードの出力位置を示すファイル・ポインターは，通常の順次出力で

はオープン時にファイルの先頭に位置づけられますが，付加順次出力では，通常ファイルの最後に位置づけられます。したがって，出力されるレコードは，通常の順次出力では既存レコードを上書きし，付加順次出力では既存レコードの追加になります。

- 1つのプログラムの中ですでに順次出力用またはアペンド用にファイルがオープンされている場合，クローズされる前にそのファイルをオープンできません。

- 順次入力用にオープンされたファイルが存在しない場合，File not found（ファイルがみつからない）エラーになります。順次出力，付加順次出力またはランダム入出力用にオープンされたファイルが存在しない場合は，新たなファイルが作成されます。

ファイル番号やレコード長などに範囲外の値を指定すると，Illegal function call（違法関数呼び出し）エラーが発生します。

注） ファイルのオープンからクローズまでの間にディスケットを取替えることはできません。取替えたディスケットにデータを書き込むとすでにあったデータは使用不可能になります。

例：

1 次の2つのステートメントは同じ機能をもち，順次出力用ファイルをオープンします。DATAというファイルが存在していた場合，既存のレコードはすべて破壊され，新しいレコードがファイルの先頭から上書きされます。

```
10 OPEN "DATA" FOR OUTPUT AS#1
```

```
10 OPEN "O",#1,"DATA"
```

既存のレコードを残しておくためには，次のように指定します。

```
10 OPEN "DATA" FOR APPEND AS#1
```

2 次の2つのステートメントは，いずれもランダム・アクセス用ファイルをオープンします。ドライブBのディレクトリーLVL1内の，PRFILE

O

ファイルをオープンします。

```
20 OPEN "B:LVL1¥PRFILE" AS 1 LEN=256
```

```
20 OPEN "RAND",1,"B:LVL1¥PRFILE",256
```

3 次のプログラムは，同じ行を 2 度印刷するために，プリンターをランダム・モードでオープンします。

```
10 FILE$="LPT1:"
20 OPEN FILE$ AS #1
30 PRINT #1,"幅80で印刷"
40 PRINT #1,"幅を255に変更する":WIDTH #1,255
50 PRINT #1,"この行にはアンダー・ラインを引く"
60 WIDTH #1,80:PRINT #1,STRING$(32,"_")
70 PRINT #1,"幅80で印刷"
Ok
```

(次のように印刷されます)

```
幅80で印刷
幅を255に変更する
この行にはアンダー・ラインを引く
幅80で印刷
```

省略時の幅 80 では，80 バイトのデータの後に改行文字が付けられます。したがって，行 30 と 40 の PRINT ではランダム・ファイルのレコード長 128 内に改行文字が含まれるため，1 行印刷するたびに改行が行われます。次に，行 40 で幅を 255 に変えています。これにより，行 50 の PRINT ではレコード長 128 に改行文字が含まれず，復帰文字により印字位置が行の先頭に移されるだけで，改行は行われません。さらに幅を 80 に戻した後，行 60 で下線を引いています。したがって，下線を引いた後は改行されます。

4 次の例は他のすべての処理によるデータの書き込みを防ぎ，順次入力モードでファイルをオープンした例です。

```
10 OPEN "TEST.DAT" FOR LOCK WRITE AS #1
20 INPUT #1,A$
30 CLOSE #1
```

ファイル・アクセスの制御は順次ファイルでも，ランダム・ファイルでもできます。複数の処理プログラムが同時に1つのファイルに対して，実行される場合はデータを失わないよう配慮してください。

5　次の例はこのプログラムによってファイルがオープンされている間，他のどの処理もこのファイルに対して読み書きできないように設定し，対象となるファイルを読み書き両方の処理を行うランダム・ファイルとしてオープンしています。

```
10 OPEN "TEST.DAT" FOR RANDOM ACCES
        READ WRITE LOCK READ WRITE AS #1
20 FIELD #1,20 AS N$
30 X$ = "DATA"
40 LSET N$ = X$
50 PUT #1,1
60 CLOSE #1
```

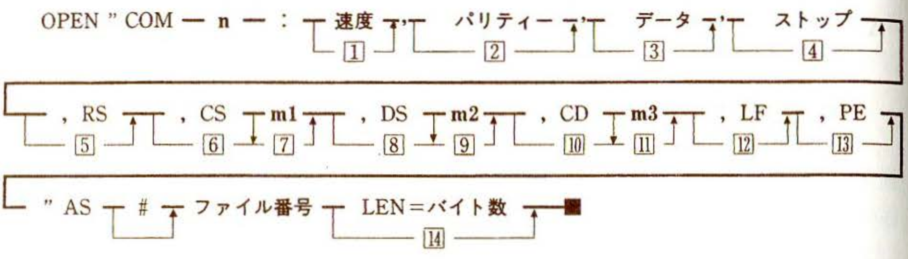
O

OPEN "COM...ステートメント

目的：

通信ファイルへの入出力ができるように，初期設定を行います。

構文：



1 300
2 E
3 8
4 2（110BPS），1（他の場合）
5 RTSを抑止しない
6 CS1000（RSが定義されたときはCS0）
7 0
8 DS1000
9 0
10 CDを制御しない
11 0
12 改行文字を送らない
13 パリティーを検査しない
14 LEN=128

説明：

nは，1か2で何枚目のASYNC通信カードを使用するかを指定します。

速度から**ストップ**までの項目は，通信データに関する指定です。次の表は，各項目ごとに上段に指定する事柄，下段に指定できる値を示しています。

速度	ビット伝送速度をビット/秒（bps）で示す整定数 110, 150, 300, 600, 1200, 2400, 4800, 9600（省略値＝300）
パリティー	送受信用パリティーを指定する 1 文字 S（Space）　：送受信ともパリティー・ビット 0 として扱う O（Odd）　：奇数パリティーで送信と受信検査を行う M（Mark）　：送受信ともパリティー・ビット 1 として扱う E（Even）　：偶数パリティーで送信と受信検査を行う N（None）　：送受信ともパリティー無しとして扱う （省略値＝E）
データ	データ・ビット数を示す整定数 7, 8（省略値＝8）
ストップ	ストップ・ビットの数を指定する整定数 1：1ビット 2：2ビット （省略値＝伝送速度が 110 の場合は 2,その他は 1）

注） これらの項目は指定位置が決められているため，この順に指定します。

RS から PE までの項目は，通信制御に関する指定です。次の表は，各項目を指定した場合の働きを示しています。

RS	RTS（送信要求）信号をオフにする
CS [m1]	CTS（送信可能）信号の検査を制御する 省略時 … RS を指定した場合　CS0 RS を使用しない場合　CS1000
DS [m2]	DSR（受信可能）信号の検査を制御する 省略時 … DS1000
CD [m3]	CD（キャリアー検出）信号の検査を制御する 省略時 … CD0
LF	各復帰文字に続けて改行文字を送る
PE	パリティー検査を行う

O

注）
- m1, m2, m3 は，信号が送られるまでの許容時間を示し，ミリ秒単位に 0 ～ 65535 の範囲で指定します。許容時間を超えても信号が送られない場合，Device Timeout（装置タイムアウト）エラーが起こります。また，許容時間に 0 を指定するか時間を省略した場合，信号の検査は行われません。
- CD は RLSD（受信回線信号検出）とも呼ばれます。
- これらの項目は，任意の順序で指定することができます。

一般に，通信ファイルに対する入力，または出力ステートメントは，CTS 信号，または DSR 信号がオフだと実行できません。CS および DS の指定は，この状態からプログラムが Device timeout エラーとなるまでの時間を決めます。また，CD 信号は通常無視されますが，これも適当な時間を指定することにより，オフ状態が続くとエラーを発生させることができます。

LF は，通信ファイルを直接にプリンターへ送る場合に使用します。これを指定すると，復帰文字（16 進 0D）の後に自動的に改行文字（16 進 0A）が付加されます。この通信ファイルを受信側で LF 付きファイルとしてオープンし，INPUT#または LINE INPUT#で読み込んだ場合は，1 レコードの読み込みが復帰文字で終わり，その後の改行文字は無視されます。

PE を指定してパリティー・エラーが検出された場合，Device I/O（装置入出力）エラーが起こり，データ・ビット数が 7 だと最上位ビットがオンになります。フレーム・エラーやオーバーラン・エラーの場合も同じ現象が発生しますが，これらのエラーは PE の指定と無関係に起こります。

ファイル番号は，通信用の入出力ステートメントでファイルを参照するために使用する番号であり，1 ～ 255 の整数式で指定します。

バイト数は，GET または PUT ステートメントで通信バッファーから読み取ることのできる，最大バイト数を指定します。省略時の値は，128 バイトです。

OPEN " COM... ステートメントは，OPEN がディスク・ファイルを割り振るのと同様の方法で，入出力バッファーを割り振ります。これ以後，他のコンピューターや周辺機器との RS-232C 非同期通信が可能になります。

通信ファイルの使用に関して，次の点に注意が必要です。

- 1つの通信ファイルを，同時に複数のファイル番号でオープンすることはできません。

- ディスケットの入出力を通信と並行に行うと，通信データが無効となる恐れがあります。したがって，ディスケット入出力を行う前に XOFF（連続コード 17）を送信して，通信を停止しておく必要があります。

- キーボード入力と通信を並行に行う場合，ビット伝送速度が速いと通信データが失われます。この場合，ビット伝送速度を 1200bps 以下に指定してください。

- BASIC は数値データを 8 ビットで表すため，数値データを送受信する場合，ストリングとして扱うのでない限り，データ・ビット数 8 で通信を開始してください。

OPEN " COM… の通信データと通信制御の指定中に間違いがある場合，Bad file name（ファイル名の誤り）エラーが起こります。ただし，どの項目の指定に間違いが有るのかを判別することはできません。
通信に関する詳細な説明は，☞ 解説編　付録 C「通信」，および操作ガイド。

例：

1
```
10 OPEN "COM1:" AS#1
```

この例では，すべて省略時の値（300bps，偶数パリティー，7 データ・ビット，1 ストップ・ビット）を使用して通信ファイルをオープンしています。

2
```
10 OPEN "COM1:2400" AS#2
```

この例は，ビット伝送速度 2400bps を指定しています。パリティー，データ・ビット数，ストップ・ビット数はいずれも省略時の値が使用されます。

3
```
20 OPEN "COM1:1200,N,8" AS#1
```

O

この通信ファイルは，ビット伝送速度 1200bps，パリティー無し，データ・ビット数 8，およびストップ・ビット数 1（省略値）でオープンされます。

4

```
10 OPEN "COM1:300,N,8,,CS,DS,CD" AS #1
```

この例では，300bps，パリティー無し，8 データ・ビット，1 ストップ・ビットの通信データを扱い，CTS，DSR，CD（RLSD）の検査は行いません。

5

```
10 OPEN "COM1:1200,,,,CS,DS2000" AS #1
```

この例では，1200bps，偶数パリティー，8 データ・ビット，1 ストップ・ビットの通信データを扱い，CTS と CD の検査は行いません。また，DSR が 2 秒以上送られない場合に，Device time out エラーとなります。

6

```
10 WIDTH "COM1:",132
20 OPEN "COM1:1200,N,8,,CS10000,DS10000,CD10000,LF" AS #1
```

この例は，通信ファイルをシリアル・プリンターの制御に使用する典型的な方法を示しています。OPEN " COM... ステートメントに LF を指定すると，各レコードの改行が自動的に行われます。

7

```
10 TRIES=6
20 ON ERROR GOTO 1000
30 OPEN"COM1:300,N,8,2,CS,DS,CD10000" AS #1
40 ON ERROR GOTO 0
50 CLOSE #1
60 OPEN "COM1:300,N,8,2,CS,DS,CD2000" AS #1
.
.
1000 TRIES=TRIES-1
1010 IF TRIES=0 THEN ON ERROR GOTO 0
1020 RESUME
.
.
```

この例では，実際に通信を開始する前にモデムの検査をしています。行 10 ～ 30 で CD のタイムアウトを 10 秒に設定してファイルをオープンし，タイムアウトとなった場合は行 1000 ～ 1020 に飛び，これを 6 回まで繰り返します。CD が検出され，モデムが正しく作動していることが確認されたら，行 50 でファイルをクローズしています。その後 CD タイムアウトを 2 秒に設定し直し，実際の通信を開始するために，行 60 で再度オープンしています。

O

OPTION BASE ステートメント

目的：　配列の添字の最小値を宣言します。

構文：

説明：　nは0か1のいずれかを指定します（配列の最小添字は0か1に限られます)。OPTION BASEを指定しない場合，省略値の0が最小の添字になります。

OPTION BASEは，配列の定義や使用より前に指定します。

例：

```
10 OPTION BASE 1
```

このプログラムで使用される配列の最小添字は，1になります。

OUT ステートメント

目的：　出力ポートへ１バイトのデータを送ります。

構文：　OUT ─ n，m ──■

説明：　n は，0 ～ 65535 の 10 進数の数式で，ポート番号（I/O アドレス）を指定します。

m はデータの連続コードを示す数式で，0 ～ 255 の範囲で指定します。

ポートから１バイトのデータを読み込むためには，INP 関数を使用します。

例：　COM1: の回線制御レジスターを変更します。

```
10 C1=&H3FB             'COM1:のポート・アドレス
20 R%=INP(C1)           '回線制御レジスターの読み込み
30 R%=R% AND (NOT 8)    'パリティー検査なしに設定
40 OUT C1,R%            '回線制御レジスターに書き込む
```

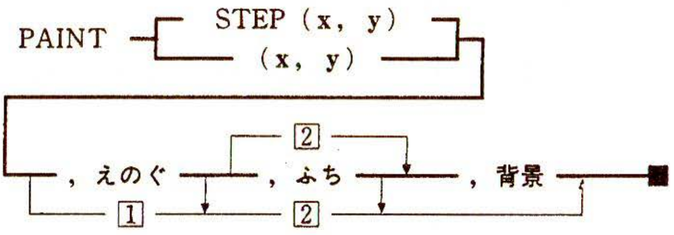
O

PAINT ステートメント

目的：

画面上の区域を指定した色（および模様）で塗りつぶします。グラフィック・モードでのみ使用できます。

構文：



1 前景色
2 **えのぐ**と同じ（**えのぐ**に模様を指定しているときは前景色）

説明：

（x,y）および STEP（x,y）は色を塗る区域内にある任意の点の座標です。STEP なしで指定した座標は絶対座標を示し，STEP 付きの座標は相対座標を示します。

えのぐには区域内を塗る色（模様）を指定します。**えのぐ**は区域内を 1 色で塗りつぶす場合と，色模様で埋める場合で指定方法が異なります。

区域内を 1 色で塗りつぶす場合

えのぐには塗りつぶす色をパレット番号で指定します。

```
10 SCREEN 1,1
20 CIRCLE(150,150),100
30 PAINT(150,150),1,3
```

区域内を色模様で埋める場合

色模様で埋めることを「彩色模様が描かれたタイルを敷きつめる」と考えます（これはタイリングと呼ばれています）。**えのぐ**にはタイルの模様を次の形式で指定します。このストリングの長さは255バイト以内です。

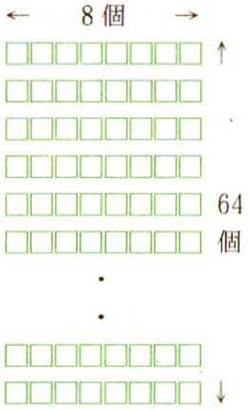
CHR$（&Hnn）+ CHR$（&Hnn）+ CHR$（&Hnn）…

このストリングは次のように記述することもできます。

CHR$（&Hnn,&Hnn,&Hnn…&Hnn）

画面上の1枚のタイルの大きさ（画面の点の数）は次のようです（縦は64個以内であればいくつでも指定できます）。

単色/16色カラー・モード



4色カラー・モード

1つの点の色を表すためには次の数のビット（画素ビット）が必要です。

＜画素ビット＞
単色 1
4色 2
16色 4

タイル模様と指定するビットの対応は次のようになります。

単色グラフィック・モード

次の例は前景色で X 模様のタイルを敷きつめる場合の指定です。8 ビットずつの対応と 16 進表示を表しています。

タイル上の模様	ビットの対応	16 進数
■□□□□□□■	10000001	&H81
□■□□□□■□	01000010	&H42
□□■□□■□□	00100100	&H24
□□□■■□□□	00011000	&H18
□□□■■□□□	00011000	&H18
□□■□□■□□	00100100	&H24
□■□□□□■□	01000010	&H42
■□□□□□□■	10000001	&H81

えのぐには各行の 16 進表示を CHR$ 関数に入れて次のように指定します。

```
CHR$(&H81,&H42,&H24,&H18,&H18,&H24,&H42,&H81)
```

4 色カラー・グラフィック・モード

次の例はパレット番号 2 の色で□模様のタイルを敷きつめる場合の指定です。

タイル上の模様	8 ビットの対応	16 進数表示
■ ■ ■ ■	10 10 10 10	&HAA
■ □ □ ■	10 10 10 10	&H82
■ □ □ ■	10 10 10 10	&H82
■ □ □ ■	10 10 10 10	&H82
■ □ □ ■	10 10 10 10	&H82
■ □ □ ■	10 10 10 10	&H82
■ □ □ ■	10 10 10 10	&H82
■ ■ ■ ■	10 10 10 10	&HAA

このように4色カラー・グラフィック・モードの場合はパレット番号0～3の色を表すために8ビット単位に区切って色を指定します。パレット番号（10進数）と対応する2ビット（2進数）は次のとおりです。

パレット番号	対応するビット
0	00
1	01
2	10
3	11

この例をタイルに指定するときは，**えのぐ**に各行の16進数表示をCHR$関数に入れて次のように指定します。

```
CHR$(&HAA,&H82,&H82,&H82,&H82,&H82,&H82,&HAA)
```

16色カラー・グラフィック・モード

16色のパレット番号は4ビットで識別できます。画面上の各1点に対して4ビット単位に行います。すなわち画面上の各1点に対応するパレット番号（0～15）を16進数（0～F）で指定します。

前述の模様でパレット番号8の地にパレット番号12の四角い模様を塗る指定は次のようになります。（□はパレット番号8，■はパレット番号12）。

タイル上の模様	16進数のパレット番号
□□□□□□□□	88888888
□■■■□■■■	8CCC8CCC
□■■■□■■■	8CCC8CCC
□■■■□■■■	8CCC8CCC

CHR$には次の16進数を指定します。

```
&H88,&H88,&H88,&H88,
&H8C,&HCC,&H8C,&HCC,
&H8C,&HCC,&H8C,&HCC,
```

&H8C,&HCC,&H8C,&HCC

このように**えのぐ**には模様を指定する値を記述します。

CHR$ (0) のみを**えのぐ**に指定することはできません。Illegall function call（違法関数呼び出し）エラーになります。背景色で塗る場合にはパレット番号 0 を指定してください。

ふちには塗りつぶす範囲を示す境界線の色をパレット番号で指定します。範囲は単色のときに 0 または 1，4 色画面のときに 0 ～ 3，16 色画面のときに 0 ～ 15 です。BASIC は指定した色に出会うまで領域を塗りつぶします。省略値は**えのぐ**に指定したパレット番号と同じです。**えのぐ**に模様を指定している場合の省略値は前景色です。

背景には，下地の色模様の中で区別の必要な行の模様を示す CHR$ 関数を指定します。すでに模様が塗られている区画に別な模様を塗る際，下地模様の中でこれから塗る模様と区別できない模様がある場合があります。このような行を，背景に指定します。ただし，区別できない行が 2 種類以上ある場合，背景には 1 行しか指定できないため，Illegal function call（違法関数呼び出し）エラーとなります。16 色カラー・グラフィック・モードでは**背景**を指定できません。

PAINT は周囲が囲まれた図形であればどのような形でも塗りつぶすことができます。ただし，複雑な形には多くのスタック空間を使用するため，プログラムの最初で CLEAR を使い，使用可能なスタック空間を増やしてください。

例：

次の例は 2 つ重なる色模様を指定した場合において，背景を指定しなかったものと，指定したものの例です。最初の模様は，背景を指定していません。そのため 2 回目の PAINT ステートメントは X 方向に 1 点列のみ描いて，実行を中断しています。次の模様は，背景に重なる模様の&H5F を指定しているので 2 回目の PAINT ステートメントは最後まで実行しています。

```
10 SCREEN 1,1 : COLOR 0,1 : LOCATE,,0
20 TIL$=CHR$(&H5F,&H5F,&H27,&H81)
30 VIEW(1,1)-(150,100),0,2
40 PAINT(125,50),CHR$(&H5F)
50 PAINT (125,50),TIL$,2
60 VIEW(160,100)-(310,198),0,2
70 PAINT(125,50),CHR$(&H5F)
80 PAINT(125,50),TIL$,2,CHR$(&H5F)
90 LOCATE 15,1,1
```

次の例は入力された色で模様を塗ります。

```
10 SCREEN 2,1 : KEY OFF : VIEW : CLS
20 LOCATE 1,13
30 PRINT "*** 16色カラー・グラフィック・モード ***"
40 LOCATE 3,1
50 INPUT "色番号(0〜15)を入力してください ==> ",N
60 DOT1$=CHR$(0,0,0,0,N,N*16+N,N,N*16+N,N,N*16+N,N,N*16+N,N,N*1
6+N,N,N*16+N)
70 DOT2$=CHR$(0,0,0,0,N,N*16,N,N*16,N,N*16,N,N*16,0,0,0,0)
80 FOR I = 0 TO 2
85 X = 120+240*I
86 Y = 250
90 CIRCLE(X,Y),120
100 IF I = 0 THEN PAINT (X,Y),N,15
110 IF I = 1 THEN PAINT (X,Y),DOT1$
120 IF I = 2 THEN PAINT (X,Y),DOT2$
130 NEXT I
```

P

PALETTE ステートメント

目的：

パレットに色を設定します。カラー・グラフィック・モードでのみ使用できます。

構文：

PALETTE ─┬─ **パレット番号，色** ─┬─■
　　　　　└──── [1] ────┘

PALETTE USING ─ **配列名** ─■

[1] ☞ 説明：

説明：

パレット番号はパレットの区画の番号を指定します。範囲は 4 色カラー・グラフィック・モードのときに 0 ～ 3，16 色カラー・グラフィック・モードのときに 0 ～ 15 です。

色はパレットに設定する色番号を指定します。範囲は 4 色カラー・グラフィック・モードのときに 0 ～ 15，16 色カラー・グラフィック・モードのときに 0 ～ 63 です。

配列名は，配列要素に色番号を持った整数配列を指定します。

パレットには次の省略時の色番号が設定されています。したがって，PALETTE ステートメントでパレットの内容を変更しない場合は，次のパレット番号に対応する色を使用することができます。

4色カラー・グラフィック・モード	
パレット番号	色番号（色）
0	0 黒
1	3 水色
2	5 紫
3	7 白

16色カラー・グラフィック・モード	
パレット番号	色番号（色）
0	0 黒
1	9 明るい青
2	18 明るい緑
3	27 明るい水色
4	36 明るい赤
5	45 明るい紫
6	54 明るい黄
7	63 明るい白
8	56 灰色
9	57 薄い青
10	58 薄い緑
11	59 薄い水色
12	60 薄い赤
13	61 薄い紫
14	62 薄い黄
15	63 明るい白

PALETTE USING ステートメントは次のような PALETTE ステートメントを簡潔に記述することができます。

```
10 '--- PALETTE USINGステートメント ---
20 A%(0)=3 : A%(1)=2 : A%(2)=1 : A%(3)=0
30 PALETTE USING A%(0)
```

```
10 '--- PALETTE ステートメント ---
20 PALETTE 0,3
30 PALETTE 1,2
40 PALETTE 2,1
50 PALETTE 3,0
```

P

PEEK 関数

目的：

指示された記憶域の位置から読み取った 1 バイトを返します。

構文：

—— PEEK（n）——

説明：

n は，読み取る記憶域のアドレスを 0 ～ 65535 の範囲で指定します。n で指定する値は，現在のセグメントの中のオフセットです。

返される値は，0 から 255 の範囲の整数です。

ビデオ・メモリーやフォント・バッファーのアドレスを指定しても返される値は保証できません。

記憶域に 1 バイトのデータを書き込むには，POKE ステートメントを使用します（☞「POKE ステートメント」）。

PLAY ステートメント

目的：　音楽を演奏します。

構文：

PLAY ― ストリング ――■

PLAY ON ――■

PLAY OFF ――■

PLAY STOP ――■

説明：　**ストリング**は次の音楽コマンドのストリング列を指定します。

AからGに、オプションの#、＋、－をつけたもの

現在のオクターブ中で，指示された音符を演奏します。AからGは音名のイからトに対応しています。すなわちハ長調の「ドレミ」はCDEに相当します。#，＋，－はAからGの文字の後ろにつけます。この番号記号（#）またはプラス記号（＋）はシャープを示し，マイナス記号（－）はフラットを示します。#，＋，－は，必ずピアノ上の黒鍵に対応していなければなりません。たとえば，B#の音符は間違った指定です。

On　オクターブ。このあとの音符のために，現在のオクターブをセットします。オクターブは7つあり，0～6の番号がついています。各オクターブはCからBまでの範囲です。オクターブ3は中央のCから始まります。省略時のオクターブは，オクターブ4です。

>n　nで指定された音符を1オクターブ上げて演奏します。nで指定された音符はオクターブ6に届くまで現時点の音階より1オクターブ上げて演奏されます。たとえば，PLAY>Aは，A音符が実行されるたびに，オクターブ6になるまで現時点より1オクターブ上げて演奏されます。

<n　nで指定された音を1オクターブ下げて演奏します。nで指定された音符は，オクターブ0に届くまで現時点の音階よりオクターブ下げて演奏されます。たとえば，PLAY<Aは，Aの音が実行されるたびに，オクターブ0になるまで現時点より1オクターブ下げて演奏されます。

P

Nn 音符 n を演奏します。n は 0 ～ 84 の範囲です。BASIC は，使用可能な 7 つのオクターブ中に，84 の音符があります。n=0 は休符です。この方法はオクターブ（On）と音名（A-G）を指定する代わりに，音符を選択する代替方法です。

Ln このあとの音符の長さを設定します実際の音符の長さは 1/n です。n は 1 から 64 の範囲です。長さと対応する音符は次のようになります。

長さ	対応する音符
L1	全音符
L2	2 分音符
L3	3 連 2 分音符の 1 つ（4 拍子の小節の 1/3）
L4	4 分音符
L5	5 分音符（1 小節の 1/5）
L6	3 連 4 分音符の 1 つ
●	
●	
●	
L64	64 分音符

ある音符だけの長さを変えるときは，音符の後に長さを続けることもできます。たとえば A16 は L16A と同じ値です。

Pn 休符の指定です。n は 1 から 64 の範囲で，L（長さ）と同じ方向で休符の長さを表わします。

（付点またはピリオド）音符の後につけ，その音符を付点音符として演奏します。すなわち，その音符の長さは 32 をかけたものになります。音符の後に 2 つ以上のドットがつくこともあり，長さはそれに応じて調整されます。たとえば，A.. は L の指定する長さの 94 倍演奏され，A... は 278 倍演奏されます。付点は休符（P）の後にも使われ，休符の長さを同様に延長します。

Tn テンポ。1 分間に演奏する 4 分音符の数を設定します。n は 32 から 255 の範囲です。省略時の値は 120 です。後述の「SOUND ステートメント」に，

一般的なテンポと，それに等しい 1 分間の拍子数の表が載せてあります。

MF　前景音楽。音楽（SOUND または PLAY で作成されたもの）は前景で実行されます。すなわち，後続の音符あるいは音は，先行の音符または音が終了するまでは開始されません。PLAY から抜け出すには Break キー機能を押します。前景音楽は省略時の状態です。

MB　背景音楽。音楽（SOUND または PLAY で作成されたもの）は，前景でなく背景で実行されます。すなわち，各音符あるいは音はバッファーに置かれ，背景で音楽を演奏している間に BASIC プログラム実行を続けることができます。通常 16 個までの音符（または休符）が背景で演奏できます。

MN　ノーマル。各音符は L（長さ）で指定された時間の 78 の間演奏されます。これは，MN，ML，および MS の，省略時の設定です。

ML　レガート。各音符は L（長さ）で指定された全期間演奏されます。

MS　スタッカート。各音符は L で指定された時間の 34 の間演奏されます。

X 変数;　特定のストリングを実行します。

これらすべてのコマンドにおいて，引数 n は，12 のような定数，または＝変数;の形です。ここで変数は変数名です。変数をこのように使用する場合および X コマンドを使用する場合には，セミコロン（;）が必要です。その他の場合，コマンド間のセミコロンは省略することができます。ただし，MF，MB，MN，ML，または MS の後にセミコロンを使うことはできません。また，ストリング中のブランクは無視されます。

変数を，＝変数;の代わりに，VARPTR$（変数）の形で指定することもできます。たとえば次のように行います。

一方法	代替方法
`PLAY"XA$;"`	`PLAY"X"+VARPTR$(A$)`
`PLAY"O=1;"`	`PLAY"O="+VARPTR$(1)`

P

Xを使うと，ストリング中に「副旋律」を記憶し，他のストリングからの違ったテンポやオクターブとつないで，繰り返し呼び出すことができます。

PLAY ONは背景音楽の割り込み処理を許可に設定します。PLAY OFFは割り込み処理を禁止にします。PLAY STOPは割り込み処理を保留に設定します。詳細は「ON PLAY (n) ステートメント」を参照してください。

例：

次の例は旋律を演奏します。

```
10 MARY$="GFE-FGGG"
20 PLAY "MB T100 O3 L8;XMARY$;P8 FFF4"
30 PLAY "GB-B-4;XMARY$;GFFGFE-."
```

次の例は，音階を演奏します。

```
10 '--- オクターブ0から6の音階を演奏
20 USCALE$="CDEFGAB":DSCALE$="BAGFEDC"
30 PLAY "O0 XUSCALE$;"
40 FOR I=1 TO 6
50 PLAY ">XUSCALE$;"
60 NEXT
70 '--- オクターブ6から0の音階を演奏
80 PLAY "O6 XDSCALE$;"
90 FOR I=1 TO 6100 PLAY "<XDSCALE$;"
110 NEXT
```

次のプログラムはキーを押して音を発生させる例です。

```
10 '+++++++++++++++++ オルガン演奏 +++++++++++++++++++
20 '+ タイプライター・キーの上２段をオルガンにします。   +
30 '+ 英数モードでタイプしてください。「ド」は「E」です。+
40 '++++++++++++++++++++++++++++++++++++++++++++++++++
50 DATA 1,2,4,5,7,8,9,-,^ ,q,w,e,r,t,y,u,i,o,p,@, [
60 DATA L16O2G#,L16O2A#,L16O3C#,L16O3D#,L16O3F#,L16O3G#,
```

```
L1603A#,L1604C#,L1604D#,L1602A,L1602B,L1603C,L1603D,L1603E,
L1603F,L1603G,L1603A,L1603B,L1604C,L1604D,L1604E
70 DIM K$(22) : DIM P$(22)
80 FOR I = 1 TO 21
90   READ K$(I)
100 NEXT I
110 FOR I = 1 TO 21
120   READ P$(I)
130 NEXT
140 '--- 音を出す
150 POKE 106,0
160 A$ = INKEY$ : IF A$ = "" THEN 160
170 FOR I = 1 TO 21
180   IF A$ = K$(I) THEN PLAY "XP$(I);"
190 NEXT
200 GOTO 150
```

P

PLAY (n) 関数

目的：　音楽背景バッファーにある音の数を返します。

構文：　── PLAY (n) ──

説明：　nは，仮引数です。音楽が前景モードで実行中のときのPLAY (n) は0を返します。返される値の最大値は32でバッファーに置いておくことのできる最大値です。

PLAY (n) は，音楽が背景 (MB) モードを使用してるときのバッファーにあるものだけを返します。

例：

```
10 '--- 背景音楽バッファーに残っている音が
20 '    5つになったとき行番号1000に飛び
30 '    他の曲を演奏する
40 PLAY "MB CDEFGAB"
50 IF PLAY(0)=3 GOTO 1000
60 GOTO 50
.
.
.
1000 PLAY "MF O4 T200 L4 MS GGGE-1"
```

PMAP 関数

目的：

ワールドの座標系とスクリーンの座標系間の座標変換を行います。グラフィック・モードでのみ使用可能です。

構文：

── PMAP（z, n）──

説明：

z は座標変換する点の座標値です。

n は，0～3 の範囲の値を指定します。0～3 の指定で返る値は次のとおりです。

0 は，ワールドの座標系をスクリーンの座標系に変換する場合の X です。
1 は，ワールドの座標系をスクリーンの座標系に変換する場合の Y です。
2 は，スクリーンの座標系をワールドの座標系に変換する場合の X です。
3 は，スクリーンの座標系をワールドの座標系に変換する場合の Y です。

PMAP は，WINDOW ステートメントによって定義されたワールドの座標系と VIEW ステートメントによって定義されたスクリーンの座標系間の座標値を変換するときに使用します。

PMAP（z, 0）と PMAP（z, 1）は，ワールドの座標系からスクリーンの座標系へ変換するのに使用します。

PMAP（z, 2）と PMAP（z, 3）は，スクリーンの座標系からワールドの座標系へ変換するのに使用します。

POINT 関数

目的：

画面上で指定した点の色を返します。
グラフィック・モードでのみ使用できます。

構文：

── POINT (x, y) ──
── POINT (n) ──

説明：

(x, y) は指定する点を絶対座標で指定します。返る値は，パレット番号です。与えられた点が範囲外だと，-1 が返されます。

n は，次の 0 から 3 の値を指定します。

この場合の POINT 関数は，グラフィック座標系において最終座標の X か Y の値を返します。

0	現在のスクリーン座標系の X 座標値を返す
1	現在のスクリーン座標系の Y 座標値を返す
2	WINDOW が有効なときに現在のワールド座標系の X 座標値を返す WINDOW が無効な場合は，現在のスクリーン座標系の X 座標値を返す
3	WINDOW が有効なときに現在のワールド座標系の Y 座標値を返す WINDOW が無効な場合は，現在のスクリーン座標系の Y 座標値を返す

関連情報については，☞「WINDOW ステートメント」「VIEW ステートメント」

例：

次の例では，点 (I, I) の現在の状態を反転します。

```
10 PSET(I,I),1-POINT(I,I)
```

次の例は，POINT 関数によって返される値を説明しています。WINDOW に沿った値の変化に注目してください。

```
10 SCREEN 2,0
20 PRINT "WINDOWが無効の場合のPOINT(N)"
30 GOSUB 110
40 WINDOW (0,0)-(719,511)
50 PRINT "WINDOWおよびSCREEN属性が有効なときのPOINT(N)"
55 GOSUB 110
60 PRINT "WINDOWおよびSCREEN属性が有効なときのPOINT(N)"
80 WINDOW SCREEN (0,0)-(719,511)
90 GOSUB 110
100 END
110 PSET (5,15)
120 FOR I=0 TO 3
130 PRINT POINT(I);
140 NEXT
150 PRINT:PRINT
160 RETURN
Ok

RUN

WINDOWが無効の場合のPOINT(N)
 5  15  5  15

WINDOWおよびSCREEN属性が有効なときのPOINT(N)
 7  496  5  15

WINDOWおよびSCREEN属性が有効なときのPOINT(N)
 7  23  5  15

Ok
```

P

POKE ステートメント

目的： 記憶域に 1 バイトのデータを書き込みます。

構文： POKE — n, m —■

説明： n はデータを書き込む記憶域のアドレスを 0 ～ 65535 の範囲で指定します。この値は DEF SEG ステートメントで定義された現在のセグメントの中のオフセットです。

m は，書き込むデータを 0 ～ 255 の範囲で指定します。

記憶域の 1 バイトを読み取るには PEEK 関数を使用します。POKE および PEEK は，データ記憶，機械語サブルーチンの読み込み，そして機械語サブルーチンとの引数および結果の受け渡しに使用します。

BASIC はアドレスに関するチェックを行わないので DOS 領域 BASIC のスタック，BASIC の変数域，および BASIC プログラム内に POKE で書き込まないように注意してください。

またビデオ・メモリーおよびフォント・バッファーには，POKE ステートメントでデータを書き込まないでください。

POS 関数

目的：　カーソルの現在の桁位置を返します。

構文：　── POS (**n**) ──

説明：　n は仮引数です。

カーソルの現在の水平方向（桁）位置を返します。返される値の範囲は，WIDTH で指定した桁数に依存します。最大値は現在の画面の有効桁数に依存します。

カーソルの垂直方向（行）位置を求めるときは CSRLIN を使います。

例：　この例は，カーソルが画面の 60 桁目を超える場合，復帰文字を PRINT します（すなわち，カーソルを次の行の始めに移動します）。

```
IF POS(0)>60 THEN PRINT CHR$(13)
```

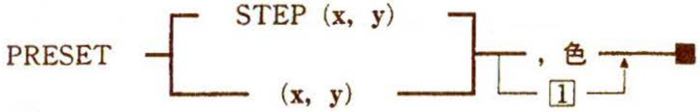
P

PRESET ステートメント

目的：

画面上の指定した座標に点（ドット）を描きます。
グラフィック・モードのみ。

構文：



[1] 背景色

説明：

(x, y) は描く点を絶対座標で指定します。

STEP (x, y) は描く点を相対座標で指定します。

色は点の色をパレット番号で指定します。

☞「PSET および PRESET ステートメント」。

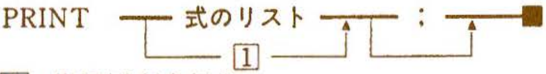
色が省略されたときに PRESET は背景色で点を描きます。すなわち，すでに点が描かれている場合は，点を消去します。PSET は，点を描くステートメントですが，PRESET は，点を消去するときに使用します。
色を指定したときの PRESET は，PSET と同様に働きます。

PRINT ステートメント

目的：

画面上にデータを表示します。

構文：



1 復帰/改行を行う

説明：

式のリストはカンマ，スペース，セミコロンで区切った，数値やストリングから成るリストです。リストの中のストリング定数はみな，引用符で囲む必要があります。

式のリストが省略されると復帰/改行のみが行われます。

PRINT ステートメントの予約語「PRINT」を入力する代わりに，「?」が使えます。「PRINT」の代わりに入力された「?」は，プログラム行の中で，「PRINT」に書き換えられます。

表示位置

各表示項目の位置は，リストの中の項目を区切るのに使う句読点で決まります。BASIC は，各行を 14 文字単位の表示区域に分割します。式のリストをカンマで区切ると，表示する値は，最初の区域の先頭から表示されます。次の値は，次の区域の先頭から表示されます。したがって，カンマを使うと，自動的にデータの先頭がそろえられます。

セミコロンで区切ると値がつめて表示されます。式の間に 1 つ以上のスペースを空けると，セミコロンで区切ったのと同じ意味になります。

カンマ，セミコロン，SPC 関数，TAB 関数で式のリストが終わると，次の PRINT ステートメントは，それらに応じてスペースをあけて同じ行から表示を始めます。カンマ，セミコロン，SPC 関数または TAB 関数で式のリストが終わらない場合は，行の終わりに復帰/改行が印刷されます。すなわち，BASIC はカーソルを次の行の始めに移動します。

表示する値の長さが，現在行の残りの文字桁数を超えると，その値は次の行の始めから表示されます。表示する値が定義された WIDTH よりも長い場合，BASIC 現在行に表示できるだけ表示し，次の物理行に値の残りの表示を続けます。表示された数のあとには常にスペースが 1 つ置かれます。また，正の数の前にはスペースが 1 つ置かれます。負の数の前にはマイナス記号が置かれます。

浮動小数点形式で表さず，7 桁以下の固定小数点形式で表しても精度の正確さが落ちない単精度数は，固定小数点または整数形式で出力されます。たとえば，10 ^ (-7) は，.0000001 と出力され，10 ^ (-8) は 1E-8 と出力されます。

設定された幅の値の桁位置に 2 バイト・コード文字がきた場合には，その行には出力されず自動的に復帰/改行を行い，次の行に 2 バイト・コード文字を出力します。

例：

```
10 X=5
20 PRINT X+5,X-5,X*(-5)
30 END
RUN
 10            0            -25
Ok
```

この例は，カンマにより，それぞれの値が次の表示区域の始めから表示されることを示しています。

```
10 INPUT X
20 PRINT X"の2乗は"X^2"そして";
30 PRINT X"の3乗は"X^3
RUN
? 9
 9 の2乗は 81 そして 9 の3乗は 729
Ok
RUN
? 21
```

```
 21 の2乗は 441 そして 21 の3乗は 9261
Ok
```

この例は，行 20 の終わりにセミコロンがあるため，両方の PRINT ステートメントが同じ行に表示されることを示しています。

```
10 FOR X=1 TO 5
20 J=J+5
30 K=K+10
40 PRINT J;K;
50 NEXT X
RUN
 5  10  10  20  15  30  20  40  25  50
Ok
```

この例は，セミコロンで区切ったので，それぞれの値が，つめて表示されることを示しています。（数のあとにはスペースが 1 つ続き，正の数の前にはスペースが 1 つ置かれます)。

P

PRINT USING ステートメント

目的：　ストリングまたは数を指定した書式で画面上に表示します。

構文：　**PRINT USING —— v$；式のリスト ——┬—；—┬——■**

説明：

v$ は，書式指定文字（下記参照）から成るストリング定数または変数です。これらの書式指定文字は，表示するデータ書式を決定します。

式のリストは，セミコロンかカンマで区切った表示データ（ストリング式，数式）です。

ストリング・フィールド

ストリングを表示するときに，表示するストリングに対応する書式指定は，次の 3 つの書式指定文字のうち 1 つを使用します。

!　　与えられたストリングの最初の 1 バイトのみを表示します。

&スペース n 個&

ストリングの左端から 2 + n バイト分の文字を表示します。スペースなしで&を指定すると，2 バイト分の文字が表示されます。スペースが 1 つだと，3 バイト分の文字が表示され，以下同様です。2 バイト・コード文字を表示するスペースは，1 バイト・コード文字の 2 個分が必要です。

ストリングがフィールドよりも長いと，余分の文字は無視されます。フィールドがストリングよりも長いと，ストリングはその欄の中で左寄せされ，右側はスペースが埋められます。

```
10 A$="LOOK":B$="OUT"
20 PRINT USING "!";A$;B$
```

```
30 PRINT USING "&  &";A$;B$
40 PRINT USING "&   &";A$;B$;"!!"
RUN
LO
LOOKOUT
LOOK OUT  !!
```

@ ストリングを可変長で表示します。@を指定するとストリングは入力どおりに表示されます。

```
10 A$="LOOK":B$="OUT"
20 PRINT USING "!";A$;
30 PRINT USING "@";B$
RUN
LOUT
```

数値フィールド

数を表示するときに，表示する数値に対応する書式指定は，次の特殊文字を使用します。

\# 番号記号（#）はそれぞれの数字の位置を示します。桁の位置は常に埋められます。表示すべき数の桁が，指定された桁の位置より少ない場合，数はその欄の中で右寄せされます（前は，スペースで埋められます）。

小数点はどこにでも挿入できます。整数部（小数点の左側）の数字を表示するように指定すると，その桁は，0も表示します。表示桁数が，表示する数値の桁数より少ない場合は，四捨五入されます。

```
PRINT USING "##.##";.78
 0.78
```

```
PRINT USING "###.##";987.654
```

P

```
987.65
```

```
PRINT USING "##.##   ";10.2,5.3,66.789,.234
10.20   5.30   66.79   0.23
```

最後の例は，書式ストリングの最後にスペースを 3 つ指定したので表示された値は分離されています。

+　書式ストリングの始めまたは終わりにプラス記号をつけると，数の符号（プラスまたはマイナス）が数の前か後ろに表示されます。

−　書式フィールドの終わりにマイナス記号をつけると，負の数は後ろにマイナス符号をつけて表示されます。

```
PRINT USING "+##.##   ";-68.95,2.4,55.6,-.9
-68.95   +2.40   +55.60   -0.90
```

```
PRINT USING "##.##-   ";-68.95,2.449,-7.01
68.95-   2.45   7.01-
```

＊＊　書式ストリングの始めに二重アステリスクをつけると，数値欄の中の先行スペースがアステリスクで埋められます。＊＊は 2 桁分の場所を確保します。

```
PRINT USING "**#.#   ";12.39,-0.9,765.1
*12.4   *-0.9   765.1
```

¥¥　二重円記号は，書式指定された数のすぐ左に，1 つの円記号を表示します。¥¥は 2 桁分を確保し，そのうち 1 つが円記号になります。指数書式を¥¥と一緒に使うことはできません。

```
PRINT USING "¥¥###.##";456.78
 ¥456.78
```

＊＊¥

書式ストリングの始めに＊＊¥をつけると，上記の2つの記号の意味を組み合わせたものになります。先行スペースはアステリスクで埋められ，数の前には円記号が表示されます。＊＊¥は，3桁を確保し，そのうち1つが円記号となります。

```
PRINT USING "**¥##.##";2.34
***¥2.34
```

,

書式ストリング中，小数点の左側にカンマを置くと，小数点の左の3桁ごとにカンマが表示されます。書式ストリングの終わりにカンマを置くと，ストリングの一部として表示されます。カンマはさらに1つの桁を指定します。

カンマは指数（^^^^）書式と共に使用できません。

```
PRINT USING "####,.##";1234.5
1,234.50
```

```
PRINT USING "####.##,";1234.5
1234.50,
```

^^^^

4つの脱字記号（アクサン・シルコンフレックス）は桁位置文字のあとに置き，指数書式で表示します。4つの脱字記号はE ± nnまたはD ± nnを表示する空間を提供します。小数点位置はどこにでも指定できます。有効数字は左寄せされ，指数が調節されます。先行＋あるいは後続＋または−が指定されなければ，小数点の左側の1桁が，スペースまたはマイナス記号を表示するのに使用されます。

```
PRINT USING "##.##^^^^";234.56
 2.35E+02
```

```
PRINT USING ".###^^^^-";-88888
.889E+05-
```

P

```
PRINT USING "+.##^^^^";123
+.12E+03
```

_ 書式ストリング中に下線があると，続く文字がリテラル文字として表示されます（その文字自身が表示されます）。

```
PRINT USING "_!##.##_!";12.34
!12.34!
```

リテラル文字自体が下線である場合は，書式ストリング内に"__"と置きます。

表示する数が指定された数値欄より大きい場合は，その数の前にパーセント記号（%）が表示されます。四捨五入によって数が欄を超えると，四捨五入された数の前にパーセント記号が表示されます。

```
PRINT USING "##.##";111.22
%111.22
```

```
PRINT USING ".##";.999
%1.00
```

指定した桁数が 24 より大きい場合は，Illegal function call（違法関数呼出し）エラーになります。

例： ストリング定数を書式ストリング内に含めるには次のように指定します。

```
PRINT USING "これは_##です。";1
これは#1です。
```

PRINT#およびPRINT# USING ステートメント

目的： データを順番にファイルへ書き込みます。

構文： PRINT# ― **ファイル番号，** ―┬― USING v$； ―┬― **式のリスト** ―■

説明： **ファイル番号**は，出力用にオープンしたファイルの番号を指定します。

v$ は，前の項「PRINT USING ステートメント」で述べた，書式指定文字をストリング式で指定します。

式のリストは，ファイルに書き込むデータ（数値，ストリング式）を指定します。

PRINT#はファイル上のデータを圧縮しません。データのイメージが，PRINT ステートメントによって画面上に表示されるのと同じ形でファイルに書き込まれます。

数式はセミコロン，カンマ，スペースで区切ります。たとえば次のようにします。

```
PRINT #1,A;B;C;X;Y;Z
```

（区切り文字にカンマを使用すると，表示欄の間に挿入される余分なスペースもファイルに書き込まれます。）

ストリング式は，区切り文字（セミコロン，カンマ，スペース）を二重引用符で囲んで指定します。

たとえば，A$=CAMERA および B$=93604-1 とします。次のステートメントは，

```
PRINT #1,A$,B$
```

P

ファイル上にCAMERA93604-1と書きます。区切り文字がありませんから，これは1つのストリングとして入力されます。2つの別々なストリングとして入力するには，PRINT#ステートメント中に次のような区切り文字を挿入してください。

```
PRINT #1,A$;",";B$
```

ファイルは次のように書かれます。

CAMERA,93604-1

これによって2つのストリング変数に読み戻すことができます。

ストリング自体がカンマ，セミコロン，有効な先行スペース，復帰文字，改行文字を含む場合は，CHR$（34）を使ってこれらを引用符で囲んでファイルに書き込んでください。

たとえば，A$="CAMERA,AUTOMATIC"およびB$="93604-1"とします。次のステートメント，

```
PRINT #1,A$;B$
```

は次のようにファイルに書きます。

CAMERA，AUTOMATIC93604-1

次のステートメント，

```
INPUT #1,A$,B$
```

は，ストリングCAMERAをA$へ，AUTOMATIC936041をB$に入力します。

これらのストリングをファイル上で正確に区切るためには，CHR$（34）を使ってファイル・イメージ上に二重引用符を書いてください。次のステート

メント，

```
PRINT #1,CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)
```

は次のようにファイルに書きます。

"CAMERA,AUTOMATIC" "93604-1"

次のステートメントは，

```
INPUT #1,A$,B$
```

"CAMERA,AUTOMATIC"を A$ へ，"93604-1"を B$ へ入力します。

PRINT#ステートメントはまた，USING オプションと共に使って，ファイルの書式を制御することもできます。たとえば次のとおりです。

```
PRINT #1,USING "¥¥###.##,";J;K;L
```

PRINT#ステートメントは，区切り文字を指定しなければなりませんが，WRITE#ステートメントは，指定せずに，ファイルに書き込めます。WRITE#ステートメントも参照してください。

例： PRINT#の例は，☞ 解説編 付録 A「ディスク入出力」

PSET および PRESET ステートメント

目的：

画面上の指定した座標に点（ドット）を描きます。
グラフィック・モードのみ。

構文：

```
PSET ─┬─ STEP (x, y) ─┬─┬─ , 色 ─┬─■
      └─ (x, y) ──────┘ └─ [1] ──┘

PRESET ─┬─ STEP (x, y) ─┬─┬─ , 色 ─┬─■
        └─ (x, y) ──────┘ └─ [2] ──┘
```

[1] 前景色
[2] 背景色

説明：

（x，y）は描く点（ドット）を絶対座標で指定します。

STEP（x，y）は描く点を相対座標で指定します。

色は点の色をパレット番号で指定します。パレット番号の範囲は画面モードによって異なります。

画面モード	パレット番号
単色	（0/1）
4 色カラー	0 ～ 3
16 色カラー	0 ～ 15

PRESET で**色**を省略すると背景色になります。この 2 つのステートメントは，このように省略時の値が異なるほかは，同じ機能です。PRESET の場合，色を省略すると背景色になるので，点を消去する場合に使用すると便利です。

範囲外の座標を PSET または PRESET に指定すると，点は描かれません。

色に範囲外のパレット番号を指定すると Illegal funtion call（違法関数呼出し）エラーになります。

例：

```
10 SCREEN 2,0
20 FOR I=0 TO 300
30 PSET (I,I)
40 NEXT
50 '--- 線を消す
60 FOR I=300 TO 0 STEP -1
70 PRESET(I,I)
80 NEXT
Ok
```

PSET で線を描いた後に，PRESET で線を消去します。

PUT ステートメント（ファイル）

目的：

ランダム・ファイル・バッファーから，ランダム・ファイルにレコードを書き込みます。また，通信バッファーから，通信ファイルへの書き込みも行います。

構文：

PUT ─┬─ # ─┬─ ファイル番号 ─┬─ , レコード番号 ─┬─■
　　 └─────┘　　　　　　　 └──── [1] ─────┘

[1] 次のレコード番号

説明：

ファイル番号は，ファイルのオープン時に指定した番号と，同じものを指定します。

ランダム・ファイルの場合，**レコード番号**は，書き込むレコードの番号を1～4,294,967,295の範囲の数式で指定します。レコード番号を省略すると，最後に書き込んだレコードの次のレコードを書き込みます。

通信ファイルの場合，**レコード番号**には，通信ファイルに書き込むデータのバイト数を指定します。この値は，OPEN "COM ステートメントのLEN指定を超えない範囲で指定します。

プログラムで，ランダム・ファイル・バッファーにデータを書くためには，PRINT#，PRINT#USING，WRITE#，LSET，RSETを使用します。ランダム・ファイルの使い方の詳細は，解説編　付録A「ディスク入出力」を参照してください。

バッファーの終わりを超えて，読み取りまたは書き込みを行おうとするとField overflow（フィールドの桁あふれ）エラーとなります。DOSは効率よく入出力を行うために，512バイトのセクター中にできるだけ多くのレコードをブロック化します。したがってPUTステートメントが実行されるたびに，いつもディスクへの物理的な書き込みを行うわけではなく，複数のPUTステートメントを実行する場合は，512バイトのセクターにまとめて書き込みます。

PUT ステートメント（グラフィック）

目的：

画面上の指定した領域に図を描きます。グラフィック・モードのみ使用できます。

構文：

PUT — （x，y），配列 ─┬─，作用 ─┬─■
**　　　　　　　　　　　　└── [1] ──┘**

[1] XOR

説明：

（x，y）は転送するイメージの，左上隅の座標を指定します。

配列は転送する情報を含む，数値配列名を指定します。この配列に関する詳細は「GET ステートメント（グラフィック）」を参照してください。

作用には，イメージの表示方法を示す，次のうちのいずれかを指定します。省略時は，XOR です。

PSET
PRESET
XOR
OR
AND

PUT は，配列にあるイメージ情報を，画面上に描きます。PUT は GET と逆の機能です。また，作用を指定することにより，これから描くイメージとすでに画面上にあるイメージとの，相互作用を持ったイメージを表示できます。

PSET は，配列のデータを画面上に表示します。

PRESET は，反転イメージ（ネガ）を生成します。生成されるイメージの画素ビットのパレット番号は，

（各画面モードで最大のパレット番号）

－（配列に持っている各画素ビットのパレット番号）

によって表すことができます。

AND，OR，XOR は，配列にある各画素ビットのパレット番号と，すでに表示されている各画素ビットとのパレット番号の論理演算を行って表示します。

AND または OR は，表示するイメージの下にすでにイメージが存在しているときに使います。XOR はアニメーションを作成するときに有効です。XOR は配列イメージ内にあるドットと同じ場所にあるドットを反転します。イメージが複雑な背景に対し 2 回 PUT すると，背景はそのまま残ります。この機能は，背景を消すことなく，対象物を動き回らせることができます。

パレット番号の論理演算（AND，OR，XOR）は，次のようになります。

OR

	単色		4色		16色											
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	3	3	5	5	7	7	9	9	11	11	13	13	15	15
2	2	3	2	3	6	7	6	7	10	11	10	11	14	15	14	15
3	3	3	3	3	7	7	7	7	11	11	11	11	15	15	15	15
4	4	5	6	7	4	5	6	7	12	13	14	15	12	13	14	15
5	5	5	7	7	5	5	7	7	13	13	15	15	13	13	15	15
6	6	7	6	7	6	7	6	7	14	15	14	15	14	15	14	15
7	7	7	7	7	7	7	7	7	15	15	15	15	15	15	15	15
8	8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
9	9	9	11	11	13	13	15	15	9	9	11	11	13	13	15	15
10	10	11	10	11	14	15	14	15	10	11	10	11	14	15	14	15
11	11	11	11	11	15	15	15	15	11	11	11	11	15	15	15	15
12	12	13	14	15	12	13	14	15	12	13	14	15	12	13	14	15
13	13	13	15	15	13	13	15	15	13	13	15	15	13	13	15	15
14	14	15	14	15	14	15	14	15	14	15	14	15	14	15	14	15
15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15

P

AND

		単色		4色												16色
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
2	0	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2
3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	0	0	0	0	4	4	4	4	0	0	0	0	4	4	4	4
5	0	1	0	1	4	5	4	5	0	1	0	1	4	5	4	5
6	0	0	2	2	4	4	6	6	0	0	2	2	4	4	6	6
7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
8	0	0	0	0	0	0	0	0	8	8	8	8	8	8	8	8
9	0	1	0	1	0	1	0	1	8	9	8	9	8	9	8	9
10	0	0	2	2	0	0	2	2	8	8	10	10	8	8	10	10
11	0	1	2	3	0	1	2	3	8	9	10	11	8	9	10	11
12	0	0	0	0	4	4	4	4	8	8	8	8	12	12	12	12
13	0	1	0	1	4	5	4	5	8	9	8	9	12	13	12	13
14	0	0	2	2	4	4	6	6	8	8	10	10	12	12	14	14
15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

XOR

	単色		4色													16色
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
4	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
6	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
7	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
10	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
12	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
13	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
14	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

また論理演算については，解説編　第２章の「論理演算子」

P

対象物のアニメーションは次のように行います。

1. 対象物を画面に PUT します（XOR を使って）。
2. 対象物の新しい位置を再計算します。
3. 古いイメージを取り除くために前と同じ位置にもう一度対象物を画面に PUT します（XOR を使って）。
4. ステップ 1 に戻ります。今度は対象物を新しい位置に置きます。

このようにして移動すると，背景は変更されません。ステップ 4 と 1 の間の時間を最小にし，ステップ 1 と 3 の間に十分な時間遅れを確保すれば，ちらつきを減少させることができます。2 つ以上の対象物のアニメーションを行う場合は，全対象物を同時に 1 ステップずつ処理しなくてはなりません。

背景を保存するのが重要でない場合は，PSET を使ってアニメーションを行うこともできます。この場合は，対象物の "前" と "後" のイメージを格納するイメージ領域が必要です。このように格納しておくと，余分の領域が効果的に古いイメージを消去します。この方法は，対象物を動かすのに 1 回しか PUT を必要としないため，上記の XOR を使った方法と比べ多少速くなります。転送するイメージが大きすぎて画面におさまらない場合，Illegal function call（違法関数呼出し）エラーになります。

例：

```
10 SCREEN 1,0
20 LOCATE 10,1,0 : PRINT "BASIC";
30 DIM A%(140)
40 GET(0,180)-(90,200),A%
50 PUT(0,180),A%
60 FOR I = 0 TO 280 STEP 5
70   PUT(I,180),A%
80   PUT(I,180),A%
90 NEXT I
100 PUT(I,180),A%
110 LOCATE,,1
```

RANDOMIZE ステートメント

目的：

乱数シード（乱数発生のための初期値）を設定します。

構文：

[1] nを求めるプロンプトが出される

RANDOMIZE TIMER

説明：

n は乱数シードとして用いる整数式を指定します。

n を省略すると，実行が一時停止され，RANDOMIZE の実行前に，次のような n を求めるプロンプトが表示されます。

```
Random number seed (-32768 to 32767)?
```

乱数シードが再設定されないと，RND 関数は，プログラムが実行されるたびに同じ乱数列を返します。プログラムの実行のたびに乱数列を変更するには，RANDOMIZE ステートメントをプログラムの始めに置いて，実行のたびに n を変えてください。

また，n の代りに TIMER 関数を使用すると実行のたびに異なる乱数シードが得られます。

例：

```
10 RANDOMIZE
20 FOR I=1 TO 4
30 PRINT RND;
40 NEXT I
RUN
Random number seed (-32768 to 32767)?
```

これに 3 と答えます。プログラムは次のように続きます。

```
Random number seed (-32768 to 32767)? 3
 .2226007  .5941419  .2414202  .2013798
Ok
RUN
Random number seed (-32768 to 32767)?
```

今度は 4 と答えます。プログラムは次のように続きます。

```
Random number seed (-32768 to 32767)? 4
 .628988  .765605  .5551561  .775797
Ok
RUN
Random number seed (-32768 to 32767)?
```

もう一度 3 と答えると，最初の実行と同じ順序の数が得られます。

```
Random number seed (-32768 to 32767)? 3
 .2226007  .5941419  .2414202  .2013798
Ok
```

次の例は，乱数発生の際，毎回異なった乱数シードが得られることを示しています。

```
10 RANDOMIZE TIMER
20 FOR I=1 TO 4
30 PRINT RND;
40 NEXT I
RUN
 .6056879  .2261304  1.704574E-03  .9517766
Ok
RUN
 .8517816  .4878492  .2165483  .3072453
Ok
```

READ ステートメント

目的：　DATA ステートメントから値を読み取り，変数に割り当てます。

構文：　READ 変数 [, 変数 ...]

説明：　**変数**は DATA ステートメントから読んだ値を受け取る数値変数，ストリング変数，配列要素のいずれかです。

READ ステートメントは，常に DATA ステートメントと組み合わせて使用します。READ ステートメントは，DATA ステートメントの値を一対一に対応させて READ ステートメント中の変数に割り当てます。READ ステートメントの変数は，数値またはストリングですが，読み取った値は，指定した変数型と一致しなくてはなりません。一致しない場合，Syntax error（構文エラー）となります。

ただし，DATA 文中の読み取られる値が，数値の場合，それを受け取る変数型は不定です。

次のプログラムは，同じ値が数値変数にもストリング変数にも同様に読み取られることを示しています。

```
10 DATA 100,-50,0,3.1416
20 READ A,B,C,D
30 RESTORE
40 READ R$,S$,T$,U$
50 PRINT A;B;C;D
60 PRINT R$;S$;T$;U$
70 END
RUN
 100 -50  0  3.1416
100-5003.1416
```

R

READ ステートメントは，複数の DATA ステートメントをアクセスできます。また，1つの DATA ステートメントを複数の READ ステートメントでアクセスすることもできます。

1つの READ ステートメントが，順に値を読んでいき，読み込む値が足りなくなると Out of data（データがない）というエラーになります。

一連の DATA ステートメントの内の，任意の行からデータを再び読み取る場合は，RESTORE ステートメントを使用してください。

例：

```
80 FOR I=1 TO 10
90 READ A(I)
100 NEXT I
110 DATA 3,08,5.19,3.12,3.98,4.24
120 DATA 5.08,5.55,4.00,3.16,3.37
```

このプログラムは，DATA ステートメントから値を配列 A に読み取ります。

REM ステートメント

目的： プログラムに注釈文を入れます。

構文： REM ─ 注釈 ──■

説明： 注釈には，通常プログラムの説明のための注釈文を，ストリングで指定します。このストリングには，漢字やひらがなも使えます。

REM ステートメントは非実行ステートメントです。プログラムをリストすると，入力時と同じように出力されます。しかし，非実行ステートメントであっても記憶域の空間を占めます。また，あまり多い注釈文は，実行時間を遅くする原因にもなります。

GOTO または GOSUB ステートメントから，REM ステートメントに分岐することもできます。実行は，REM ステートメント後の最初の実行可能ステートメントから続けられます。

注釈は，:REM の代わりにアポストロフィー（'）を用いて行の終わりに付け加えることもできます。REM またはアポストロフィーの後は，非実行ステートメントと見なされるので，注釈は，行の中の最後のステートメントに設定されなければなりません。

例：

```
100 REM 合計を求める
110 SUM=0:REM SUMの初期化
120 DATA 1,2,3,4,5,6,7,8,9,10
130 FOR I=1 TO 10
140 READ V(I)
150 SUM=SUM+V(I)
160 NEXT I
```

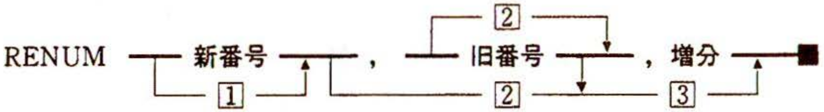
110は次のように書くこともできます。

```
110 SUM=0      'SUMの初期化
```

RENUM コマンド

目的： プログラムの行番号を付け替えます。

構文：



[1] 10
[2] プログラムの最初の行
[3] 10

説明：

新番号には，新しくつける行番号の，最初の行番号を指定します。省略時の値は 10 です。

旧番号には，行番号の付け替えを開始する。現在のプログラム中の行番号を指定します。省略時の値は，プログラムの最初の行の行番号です。

増分は新しく付ける行番号の増分です。省略時の値は 10 です。

RENUM を行うと，GOTO，GOSUB，THEN，ON … GOTO，ON … GOSUB，RESTORE，RESUME，ERL ステートメントに続く，すべての行番号を参照し，新しい行番号と対応するように変更します。これらのステートメントの後に，存在しない行番号があった場合，Undefined Line number xxxxx in yyyyy（yyyyy 中の xxxxx 行番号が不定）というエラー・メッセージが表示されます。誤った行番号の参照（xxxxx）は RENUM によって変更されませんが，行番号 yyyyy は変更されます。

注） RENUM はプログラム行の順序を変える（たとえば，プログラムが 10,20,30 という 3 つの行番号を持っているとき，RENUM 15,30 とする）こと，または 65529 より大きな行番号を作成することはできません。どちらも Illegal function call（違法関数呼出し）エラーが起こります。

例：

`RENUM`	プログラム全体の番号を付け替えます。最初の新しい行番号は 10 です。行の増分は 10 です。
`RENUM 300,,50`	プログラム全体の番号を付け替えます。最初の新しい行番号は 300 です。行の増分は 50 です。
`RENUM 1000,900,20`	行番号 900 以降の行番号を，行番号 1000 から増分 20 で付け替えます。

RESET コマンド

目的：　すべてのディスク・ファイルをクローズし，システム・バッファーを消去します。

構文：　RESET ──■

説明：　オープン・ファイルがすべてディスク上にある場合，RESETは，ファイル番号のついていないCLOSEを実行することと同じになります。

RESTORE ステートメント

目的：

READ ステートメントで，再度 DATA ステートメントを読むために，DATA ステートメントの行番号を指定します。

構文：

[1] 最初の DATA ステートメントの行

説明：

行は，プログラムにある DATA ステートメントの行番号を指定します。

行を指定すると，次の READ ステートメントは，指定した行の，DATA ステートメントのデータを読み取ります。

行を省略すると，次の READ ステートメントは，1 番最初の DATA ステートメントのデータを読み取ります。

例：

```
10 READ A,B,C,D,E,F
20 RESTORE 50
30 READ G,H,I
40 DATA 57,68,79
50 DATA 80,91,102
60 PRINT A;B;C;D;E;F;G;H;I
RUN
 57  68  79  80  91  102  80  91  102
Ok
```

行 20 の RESTORE ステートメントで，行 50 のデータを参照するように設定し直したので，行 30 で読み取られる値は，それぞれ 80,91,102 となります。

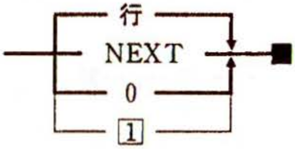
RESUME ステートメント

目的：

エラー回復手順が行われた後，プログラム実行を再び続けます。

構文：

RESUME



[1] 0

説明：

RESUME の構文は，どこから実行を再開するかによって，次のように異なります。

RESUME または RESUME 0

実行は，エラーを起こした行から再開されます。

注）RESUME 0 ステートメントを含むプログラムの番号付け替えをしようとすると Undefined Line number（行番号が不定）エラーになります。RESUME 0 はそのままですから，問題はありません。

RESUME NEXT

実行は，エラーを起こした行のすぐ後のステートメントから再開されます。

RESUME 行

実行は，指定された行番号から再開されます。

エラー処理ルーチンの外に RESUME ステートメントがあると，RESUME without error（エラーのない RESUME）というメッセージが出されます。

例：

```
10 ON ERROR GOTO 900
.
.
.
900 IF (ERR=230)AND(ERL=90)THEN PRINT "もう一度":RESUME 80
```

行 900 がエラー処理ルーチンの始まりです。行 90 でエラー 230 が起こったとき，プログラムは RESUME により，行 80 に戻ります。

R

RETURN ステートメント

目的：

サブルーチンから戻ります。「GOSUB と RETURN ステートメント」も参照してください。

構文：

[1] 制御を受けた次の行

説明：

行は，プログラムの戻る行番号です。

割り込み処理したサブルーチンから，RETURN ステートメントで戻る場合，行番号を指定しないと，割り込み処理の起きた場所に戻ります。

RETURN 行は割り込みが起きた行を無視して戻る場合に，有効です。この場合，割り込み処理時に実行されていた GOSUB，WHILE，FOR は，いずれも有効で，割り込みから戻るとこれが続行されるため，割り込み発生箇所以外に戻る場合は注意が必要です。

RIGHT$ 関数

目的：

ストリング x$ の右端から n バイト分の文字を返します。

構文：

── RIGHT$(x$, n) ──

説明：

x$ は，ストリング式を指定します。

n は，返すストリングの長さを整数式で指定します。1 バイト・コード文字は，文字数とバイト数が一致しますが，2 バイト・コード文字（漢字・ひらがな，など）は，2 バイトで 1 文字です。

n が x$ の長さより大きいか，等しい場合は，x$ が返されます。n がゼロの場合，長さゼロのストリングが返されます。

2 バイト・コード文字を含む x$ で 2 バイト・コード文字の右側 1 バイトの長さまでしか n を指定しなかったときは，該当位置に，2 バイト・コード文字の内部コードの下位バイトが設定されます。したがって，コードに対応する文字が異なりますので n の指定には注意が必要です。

例：

次の例はストリング A$ の右端から 3 文字を返します。

```
10 A$="神奈川県　横浜市"
20 PRINT RIGHT$(A$,6)
RUN
横浜市
Ok
```

R

次のユーザー定数関数 FNKRIGHT$ は，ストリングの右端から n 文字を返します。

```
10 DEF FNKRIGHT$(X$,N)=MID$(X$,-KPOS(X$,-(KLEN(X$)-N)
                *(KLEN(X$)>N)+1)*(N<>0)-255*(N=0))
20 X$="神奈川県　川崎市"
30 PRINT FNKRIGHT$(X$,3)

RUN
川崎市
Ok
```

RMDIR コマンド

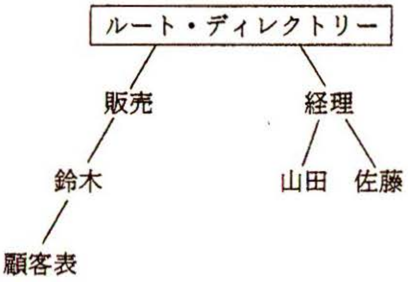
目的： ディスク上のディレクトリーを削除します。

構文： RMDIR — パス名 ——■

説明： **パス名**は削除するディレクトリーを 63 文字以内のストリング式で指定します。**パス名**の指定については，☞ 解説編　第 3 章の「ファイル指定とパス」および「ディレクトリー」

ディレクトリーにファイルが登録されている場合は，そのディレクトリーを削除できません。Path/file access error（パス指定の誤り）になります。

例： 現在次のようなディレクトリーがあると想定します。



ルート・ディレクトリーにいて**報告書**というディレクトリーを削除する場合はこのようなパス名を指定します。

```
RMDIR "販売¥鈴木¥報告書"
```

現行ディレクトリーを**経理**にし，ディレクトリー**山田**を削除します。

```
CHDIR "経理"
RMDIR "山田"
```

ディレクトリー**山田**を削除するには現行ディレクトリーをルート・ディレクトリーにし，削除するためのパスを指定する方法もあります。

```
CHDIR "¥"
RMDIR "経理¥山田"
```

ルート・ディレクトリーおよび現行ディレクトリーは削除することができません。また，現行ディレクトリーが**山田**でディレクトリー**経理**を削除しようとすると，Path/file access error となります。同じく，KILL コマンドをディレクトリー削除のために使用しようとすると，Fail not found となります。

RND 関数

目的： 0と1の間の乱数を返します。（0≦RND（x）＜1）

構文：

```
── RND ─┬─ (x) ─┬──
        └────────┘
```

説明： xには新規の乱数シードを設定するための数式を指定します。
RND関数は，乱数シードを再設定しないとプログラム実行のたびに同じ乱数列が生成されます。再シードはRANDOMIZEステートメントを使って行うのが最も簡単です。

指定する数式が

負の場合……新規の乱数を発生します。
0の場合……前回と同じ乱数を発生します。
正または省略したとき……乱数列において前回の次の乱数を発生します。

したがって，異なる乱数シードを得るためには，xの指定を毎回変える必要があります。

範囲が0（ゼロ）からnまで（0,nを含む）の乱数を得るためには，次の式を使用してください。

INT（RND＊（n＋1））

例：

```
10 FOR I=1 TO 3
20 PRINT RND(I);     '--- X>0
30 NEXT I
40 PRINT:X=RND(-6) '--- X<0
50 FOR I=1 TO 3
60 PRINT RND(I);     '--- X>0
70 NEXT I
80 RANDOMIZE 856     '--- 乱数シードの設定
90 PRINT:X=RND(-6) '--- X<0
```

```
100 FOR I=1 TO 3
110 PRINT RND;        '--- X>0
120 NEXT I
130 PRINT:PRINT RND(0)
RUN
 .1213501  .651861  .8688611
 .9045178  .7123311  .2700312
 .9045178  .7123311  .2700312
 .2700312
Ok
```

結果の最初の行は，正の x を使って生成した 3 つの乱数を表示しています。

行 40 は，乱数発生ルーチンを再シードするために負の数を指定しました。このシード後に生成された乱数は結果の 2 行目です。

行 80 は，乱数発生ルーチンを RANDOMIZE ステートメントを使って再シードしました。行 90 は，行 40 で使用したものと同じ負の数を使って RND を呼び出し，もう一度再シードします。これによって RANDOMIZE ステートメントの効果は取り消されます。結果の 3 番目の行は 2 行目と同じになっています。

行 130 の RND は引数 0 を指定しました。したがって，表示された最後の数はその前の数と同じです。

RSET ステートメント

目的：

ランダム・ファイル・バッファー内にデータを移行します。これは，PUT ステートメントで，ランダム・ファイルにデータを書き込むための準備です。

構文：

RSET ─ ストリング変数＝x$ ──■

説明：

ストリング変数は FIELD ステートメントで定義した変数名を指定します。x$ には，移行するデータをストリング式で指定します。

詳しい説明は，☞「LSET および RSET ステートメント」

R

RUN コマンド

目的：

プログラムの実行を開始します。

構文：

RUN ─┬─ **行** ─┬─■
　　　 └─ [1] ─┘

[1] 最小の行番号

RUN ─ **ファイル指定** ─┬─ **, R** ─┬─■
　　　　　　　　　　　 └─ [2] ─┘

[2] ファイルをすべてクローズする

説明：

行には実行を開始する記憶域にあるプログラムの行番号を指定します。

ファイル指定には，「装置名：」または「ドライブ指定子：」形式のストリング式で，実行するファイルを指定します（ファイル名については，☞ 解説編　第3章「ファイル指定とパス」）。

装置名またはドライブ指定子を省略すると，省略時ドライブの現行ディレクトリーが参照されます。また，パス名を指定することもできます。
ファイル指定に間違いがある場合，Bad file name（ファイル名の誤り）エラーとなり，読み込みは取り消されます。エクステンションを省略すると，.BAS が参照されます。

RUN または RUN **行**

現在記憶域にあるプログラムを実行します。行を指定すると，指定した行番号から実行します。行を省略すると，最小の行番号から実行を開始します。

RUN **ファイル指定**

ファイルをディスクから記憶域に読み込んで実行します。この場合，指定し

たプログラムを読み込む前に，オープンしているファイルをすべてクローズし，現在の記憶域の内容を削除します。しかし，R オプションを使うと，すべてのデータ・ファイルはオープンされたままです。☞ 解説編　付録「ディスク入出力」

RUN コマンドを実行すると，音楽は前景に設定し直されます。

例

```
10 PRINT 1/7
RUN
 .1428572
Ok
10 P1=3.141593
20 PRINT P1
Ok
RUN 20
 0
Ok
```

R

SAVE コマンド

目的：

記憶域にある BASIC プログラムをディスクに保管します。

構文：

```
                     ┌─ , A ─┐
SAVE ─ ファイル指定 ─┼─ , P ─┼─■
                     └─ [1] ─┘
```

[1] 圧縮 2 進形式

説明：

ファイル指定には，「装置名：」または「ドライブ指定子：ファイル名」の形式のストリング式で，保管するファイルを指定します（ファイル名については，☞ 解説編 第 3 章「ファイル指定とパス」）。

装置名またはドライブ指定子を省略すると，省略時ドライブの現行ディレクトリーが参照されます。また，パス名を指定することもできます。
ディスク・ファイルの場合，ファイル名の長さが 8 バイト以下で，エクステンションが与えられていないと，.BAS が名前に付け加えられます。ディスク上に，すでに同じファイル名を持つファイルがあるとファイル内容が，これから保管するものに書き換えられます。

選択肢の A を指定すると，プログラムは文字形式（内部コード形式）で保管されます。

選択肢の P を指定すると，プログラムはコード化 2 進形式で保管されます。

選択肢を省略すると，プログラムは圧縮 2 進形式で保管されます。

文字形式ファイルは，圧縮 2 進形式ファイルより多くの空間を必要としますが，マージするファイルは，文字形式で保管しなければなりません。

コード化 2 進形式によるプログラムの保管は，プログラム保護のための機能です。保管されたプログラムを変更することはできません。プログラムも記憶域に読み込んでも，LIST や EDIT を実行しようとすると Illegal function call（違法関数呼び出し）エラーになります。

注)

- コード化2進形式で保管されたファイルを，再度修正することはできません。この保護を解除する手段はありませんので注意してください。

- 保管されるファイルは，どのような形式であってもファイル名の長さが8バイト以下で，エクステンションがついていなければ .BAS をつけます。BASIC は，ファイル管理において，保管された形式の識別を行っていません。したがって，ファイル指定をするときに後で識別できるような名前をつけてください。

例：

`SAVE "INVENT"`	記憶域のプログラムを INVENT というファイル名で保管します。プログラムは，DOS 省略時ドライブ内のディスクに圧縮2進形式で保管されます。ファイル名には，.BAS がつけられます。
`SAVE "B:PROG",A`	プログラム PROG.BAS を文字形式で保管します。
`SAVE "A:SECRET.BOZ",P`	プログラム，SECRET.BOZ をコード化2進形式で保管します。このファイルは変更できません。

S

SCREEN 関数

目的：

画面上の，指定した行および列にある文字の連続コードや属性を返します。

構文：

```
── SCREEN ( 行, 桁 ─┬─ , z ─┬─ ) ──
                    └────────┘
```

説明：

行は行を 1 ～ 24 の範囲で指定します。WIDTH で任意の行範囲を指定した場合の有効範囲はその範囲に依存します。

桁は桁を 1 ～ 80 の範囲で指定します。WIDTH で任意の行範囲を指定した場合の有効範囲はその範囲に依存します。

z には連続コードを返すときに偽（0）を，色や属性を返すときに真（-1）を指定します（この場合の真は 0 以外であれば有効です）。

返る連続コードと文字の対応は，☞ 解説編 付録 D「コード体系」。

全角文字の右半分または左半分だけを指定した場合でも，その全角文字のコードや属性が完全な形で返されます。

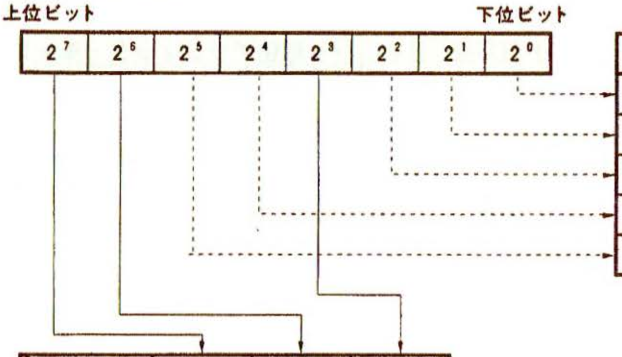
z が真のときに返される値の範囲は 0 ～ 255 で，単色モード/カラー・モードそれぞれ表に示す属性の重みを足し合わせたものです。

単色モード

属性	重み
全角文字	1
全角文字の 2 バイト目	2
反転	4
高輝度	8
縦罫線	16
横罫線	32
下線づけ	64
明滅	128

単色グラフィック・モードの場合，高輝度，明滅の属性は意味を持ちません。

カラー・モード



属　性	重み
全角文字	1
全角文字の 2バイト目	2
反　　転	4
縦 罫 線	16
横 罫 線	32

重み / 色(番号)		128(青)	64(緑)	8(赤)
黒	0	1	1	1
青	1	0	1	1
緑	2	1	0	1
水色	3	0	0	1
赤	4	1	1	0
紫	5	0	1	0
黄	6	1	0	0
白	7	0	0	0

パレット番号 0 〜 7 までの色を青，緑，赤で表現すると前記のような構成になります。色の重みは，表示されている文字がパレット番号 8 〜 15 の色であっても 0 〜 7 までの色として表現されます。

色属性を返す場合返る値を V とすると，文字の色番号は次の式で求められます。

C=−（(V AND 128）=0）*1−（(V AND 8）=0）*4

C は 0 〜 7 までの色番号となります。

グラフィック・モードにおいて，指定した位置に図形情報（点または線）が含まれていると，SCREEN 関数は図形情報が書かれる前の連続コードを返します。

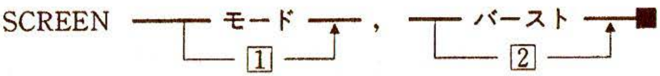
例：

```
10 SCREEN 2,0 '--- 単色グラフィック・モード
20 LOCATE 10,10
30 PRONT "ABC"
40 X=SCREEN(10,10) '--- Aの連続コードを返す
50 PRINT X
RUN
 65
Ok
```

SCREEN ステートメント

目的： 画面モードを設定します。

構文：



[1] [2] 前の値

説明： **モード**は文字モード/グラフィック・モードを0～2で指定します。

モード	画面
0	文字モード
1	グラフィック・モード
2	

バーストは単色モードまたはカラー・モードの指定を0または1で行います。

モードと**バースト**の組み合わせにより設定できる画面モードは，ハード・ウェア・システムにより異なり，次のようになります。

システム別画面モード

画面	単色	16ドット・カラー	24ドット・カラー
0，0	単色文字	単色文字	単色グラフ
0，1	単色文字	カラー文字	カラー文字
1，0	単色グラフ	単色グラフ	単色グラフ
1，1	単色グラフ	4色カラー・グラフ	16色カラー・グラフ
2，0	単色グラフ	単色グラフ	単色グラフ
2，1	単色グラフ	16色カラー・グラフ	16色カラー・グラフ

注） 旧Cモデルのみ2，1のときに単色グラフィック・モードになります。

各画面モードで表示できる文字の数は次のとおりです。

文字の数

モード	文字数（横× 縦）
文字モード	80×25
4 色カラー・グラフィック・モード	40×25
単色/16 色カラー・グラフィック・モード	78×25

グラフィック・モードの点の数は次のとおりです。

システム別グラフィック・モードの点の数

画面	16 ドット単色	16 ドット・カラー	24 ドット単色	24 ドット・カラー
単色	720×512	720×512	1024×768	1024×768
4 色カラー	-	360×512	-	-
16 色カラー	-	720×512	-	1024×768

例：

次のプログラムは 16 色カラー・グラフィック・モードの使用例です。

```
10 '++++++++++++++++++++++++++++++++++
20 '+***色をつけた線を描くプログラム***+
30 '+ カーソル移動キーで線を描くことが +
40 '+ できます。                       +
50 '+ 英数モードでDATAステートメントに +
60 '+ 記述している英字をタイプすると色 +
70 '+ を変更できます。                 +
80 '++++++++++++++++++++++++++++++++++
90 '--- 画面の初期設定
100 SCREEN 2,1 : LOCATE,,0 : KEY OFF : CLS
110 DATA b,c,g,i,l,m,o,p,r,s,t,u,v,w,y
120 DATA 11,62,18,1,38,21,52,12,36,19,43,8,29,63,6
130 DIM CNAM$(15),COL%(15) : C=14
140 FOR I = 1 TO 15
```

```
150   READ CNAM$(I)
160 NEXT I
170 FOR I = 1 TO 15
180   READ COL%(I)
190   PALETTE I,COL%(I)
200 NEXT I
210 '--- キー入力の処理
220 POKE 106,0
230 A$ = INKEY$ : IF A$ = "" THEN 230
240 IF LEN(A$) <> 2 THEN 300
250 K$ = MID$(A$,2,1)
260 IF K$ = CHR$(72) THEN DRAW "C=C;U20"
270 IF K$ = CHR$(75) THEN DRAW "C=C;L20"
280 IF K$ = CHR$(77) THEN DRAW "C=C;R20"
290 IF K$ = CHR$(80) THEN DRAW "C=C;D20"
300 FOR I = 1 TO 15
310   IF A$ = CNAM$(I) THEN C = I
320 NEXT I
330 GOTO 210
```

S

SGN 関数

目的：　数式 x の符号を返します。

構文：　── SGN（**x**）──

説明：　x は，符号を調べる数式を指定します。

SGN（x）は，

- x が正なら，SGN（x）は 1 を返します。
- x がゼロなら，SGN（x）は 0 を返します。
- x が負なら，SGN（x）は-1 を返します。

例：

```
ON SGN(X)+2 GOTO 100,200,300
```

は，x が負なら 100 へ，x がゼロなら 200 へ，x が正なら 300 へ分岐します。

SHELL ステートメント

目的： DOS コマンド・レベルで実行可能なコマンドを実行します。

構文： **SHELL — ストリング —■**

説明： **ストリング**は，実行するコマンドをストリングで指定します。

SHELL は，DOS のコマンド処理プログラム（COMMAND. COM）をBASIC の下で実行し，与えられたコマンド（ストリング指定されたもの）を処理します。コマンドは DOS のコマンド・レベルで実行可能なコマンドであれば，DOS のモードと同じように処理することができます。したがって SHELL は，BASIC の下で，BASIC 以外のプログラム（.EXE や，.COM など）を実行することができます。

ストリングに実行コマンドとしてプログラム名を指定する場合，エクステンションがないと，プログラムは，.COM ファイル，.EXE ファイル，.BAT ファイルの順で検索されます。**ストリング**によって指定されたプログラム・ファイルが見つからないときは，エラーになります。

プログラム・ファイルの場合ファイル名と少なくとも 1 つのブランクによって区切られた文字は，プログラムのパラメーターとみなされます。

ストリングを省略すると DOS のコマンド・レベルになります。この DOS のモードを終了させて，BASIC のモードに戻るには，DOS のコマンド EXIT を入力します。

SHELL ステートメントで実行する，.COM や，.EXE のプログラムは，「子プログラム」と呼ばれます。「子プログラム」は，実行されている間中，ずっと記憶域内に存在します。「子プログラム」の実行が終了したときは，BASIC をそのまま続けることができます。ただし，そのまま BASIC を続けるだけの領域が記憶域にないときは，Can't continue after SHELL（SHELL ステートメントの後の実行ができない）というエラーになります。この場合すべてのファイルはクローズされ，BASIC から DOS

のモードへ戻ります。

BASICに戻ったときは，次のような状態になっています。

1. BASICの画面モードは，DOSの画面モードに依存しています。
2. カーソルの位置は，DOSのEXITコマンドを実行した位置にとどまります。
3. パレットの色は省略時パレットの色に初期化されます。

SHELLステートメントの下でさらにBASICを呼ぶことはできません。実行しようとするとYou can not SHELL to BASIC（子プログラムとしてのBASICを実行しようとした）というエラーになります。また，SHELLステートメントで実行する「子プログラム」を呼び込むだけの領域が記憶域にない場合はOut of memory（記憶域を超えた）というエラーになります。

DOSをディスケットから始動し，SHELLを実行する場合は次の点に注意してください。

- システム・ディスケットがあるドライブを開けたままSHELLを実行すると**作動不可エラー**のメッセージが表示されます。この場合は，必ずシステム・ディスケットを始動したドライブに正しく設置し，**R（再試行）を選択してください**。ここでA（中止）またはI（強行）を選択すると中止または強行できない場合があります。

- DOSを始動したドライブにシステム・ディスケット以外のデータ・ディスケットが入っているとFile not found（ファイルが見つからない）というメッセージが表示されます。この場合もシステム・ディスケットをDOSを始動したドライブに正しく設置した後，再び実行してください。

例：

```
SHELL "DIR"
```

は，DOSのモードにおいてDOSコマンドを指定しているのと同様にディレクトリーにあるすべてのファイルに関する情報を表示します。

SIN 関数

目的：　角度の正弦を返します。

構文：　── SIN（x）──

説明：　x はラジアンで表した角度を，数式で指定します。

度をラジアンに変更するためには，3.141593/180 を掛けます。

この関数は，単精度で計算した値を返します。倍精度で計算するためには BASIC を始動するときに選択肢/D を指定します。

例：　次のプログラムは，度をラジアンに変更した後，90 度の正弦を計算しています。

```
10 PI=3.141593
20 DEGREES=90
30 RADIANS=DEGREES*PI/180
40 PRINT SIN(RADIANS)
RUN
 1
Ok
```

S

SOUND ステートメント

目的：　スピーカーから音を発生します。

構文：　**SOUND — 周波数，持続時間 —■**

説明：　**周波数**は発生させる音を Hz（ヘルツ）で指定します。値は 37 ～ 32767 の範囲の数式です。

持続時間は周波数で指定した音を発生し続ける時間を 55 ミリ秒きざみで指定します。値は 0 ～ 65535 の範囲の数式です。実際に音が発生している時間は，**持続時間**× 55 ミリ秒です。音の発生は 65535 × 2.5 ミリ秒が最長であるため，55 ミリ秒きざみのときに**持続時間**に 2979 以上の値を指定しても音の発生時間は変りません（きざみは，システム環境により多少変動することがあります)。

音は音楽バッファーに入っているので音が発生している間に，次の SOUND ステートメントに出会った場合，前回の音の発生が完了してから，次の SOUND ステートメントの音を発生します。

チューニング・ノート（基準音）A は，周波数 440Hz です。次の表は，音名とその周波数の関係を，中央の C の両側 2 オクターブにわたって表したものです。

音　名	周波数	音　名	周波数
C	130.810	C	523.250
D	146.830	D	587.330
E	164.810	E	659.260
F	174.610	F	698.460
G	196.000	G	783.990
A	220.000	A	880.000
B	246.940	B	987.770
C*	261.630	C	1046.500
D	293.660	D	1174.700
E	329.630	E	1318.500
F	349.230	F	1396.900
G	392.000	G	1568.000
A	440.000	A	1760.000
B	493.880	B	1975.500

* 中央のC。ある音に対して，1オクターブ高い（低い）音を得るには，その周波数を2倍（半分）にすると近似的に求められます。

無音時間を作るには，SOUND 32767，持続時間の形式で指定してください。

1拍子の持続時間は，1092を1分間の拍子数で割ることによって求めます。

S

次表は，代表的なテンポを示しています。

テンポ		１分当りの拍子数	１拍子当りのきざみ数
きわめて遅い	Larghissimo		
●	Largo	40-60	27.30-18.20
●	Larghetto	60-66	18.20-16.55
●	Grave		
●	Lento		
●	Adagio	66-76	16.55-14.37
遅い	Adagietto		
●	Andante	076-108	14.37-10.11
中くらい	Andantino		
●	Moderato	108-120	10.11- 9.10
速い	Allegretto		
●	Allegro	120-168	9.10- 6.10
●	Vivace		
●	Veloce		
●	Presto	168-208	6.50- 5.25
きわめて速い	Prestissimo		

例：

次のプログラムはグリッサンド演奏（低い音から高い音へまたは高い音から低い音へすべるように演奏すること）を行います。

```
10 FOR I=440 TO 1000 STEP 5
20 SOUND I,0.5
30 NEXT
110 FOR I=1000 TO 440 STEP -5
120 SOUND I,0.5
130 NEXT
RUN
Ok
```

SPACE$ 関数

目的：　n 個のスペースをストリングで返します。

構文：　—— SPACE$ (**n**) ——

説明：　n はスペースの数を 0 ～ 255 の範囲で指定します。

例：

```
10 FOR I=1 TO 5
20 X$=SPACE$(I)
30 PRINT X$;I
40 NEXT I
RUN
  1
   2
    3
     4
      5
Ok
```

このプログラムは，それぞれの数値の前に，数値の数分のスペースを置く指定をしています。ただし，数値を表示する場合，正の数の前には，スペースが 1 つ置かれます。

S

SPC 関数

目的：

n 個のスペースを出力します。

構文：

―― SPC (n) ――

説明：

n はスペースの数を 0 ～ 32767 の範囲で指定します。

n が装置の定義された幅より大きい場合，値は n MOD 幅 となります。SPC は，必ず PRINT，PRINT USING，LPRINT，LPRINT USING PRINT # ステートメントと共に使用します。単独で使用することはできません。

SPC 関数で指定したスペースがデータの終わりにあると，スペースの後にセミコロンを指定したのと同じように復帰/改行は付け加えられません。

例：

```
PRINT "OVER"SPC(15) "THERE"
OVER               THERE
Ok
```

この例は，OVER と THERE を，15 スペース隔てて表示します。

SQR 関数

目的： 平方根を返します。

構文： —— SQR (x) ——

説明： x は，0 以上の数式を指定します。この関数は，倍精度で計算した値を返します。

例：

```
10 FOR X=10 TO 25 STEP 5
20 PRINT X,SQR(X)
30 NEXT
RUN
 10            3.162278
 15            3.872984
 20            4.472136
 25            5
Ok
```

この例は，数 10,15,20,および 25 の平方根を計算します。

S

STOP ステートメント

目的： プログラムの実行を停止し，コマンド・レベルに戻ります。

構文： STOP ——■

説明： STOP ステートメントは，それまでのプログラム状態のままで，実行を停止します。STOP は，プログラムの中のどこでも使用できます。プログラム実行中に STOP ステートメントが実行されると，次のメッセージが表示されます。Break in nnnnn（ここで nnnnn は STOP が実行された行番号です。）END ステートメントと違って，STOP ステートメントはファイルをクローズしません。STOP の実行後は，常にコマンド・レベルに戻ります。

CONT コマンドは停止したプログラムの実行再開ができるので，STOP ステートメントと組み合わせて使用すると，プログラムのデバックに有用です。この例は，STOP と CONT を使用して，プログラム実行途中の変数の内容を調べています。

例：

```
10 INPUT A,B
20 TEMP=A*B
30 STOP
40 FINAL=TEMP+200:PRINT FINAL
RUN
? 26,2.1
Break in 30
Ok
PRINT TEMP
 54.6
Ok
CONT
 254.6
Ok
```

STR$ 関数

目的： xの値のストリング式を返します。

構文： —— STR$ (**x**) ——

説明： **x** は，ストリングに変換する数式を指定します。

変換の対象となるものは，連続コードの 43 (+)，45 (−)，46 (.) と数字に対応する 48 ～ 57 です。x が正の場合，STR$ の返すストリングは先行スペース（プラス記号のために空けられているスペース）が含まれます。たとえば次のようになります。

```
PRINT STR$(321);LEN(STR$(321))
 321 4
Ok
```

ストリングを数値にするには，VAL 関数を使用します。

例： 入力される数の桁数によって別々の処理を行うときは，次のように行います。

```
10 INPUT "数を入力してください";N
20 ON LEN(STR$(N))-1 GOSUB 30,100,200,300
```

S

STRING$ 関数

目的：

連続コード m の文字を n 個並べたストリング，または x$ の最初の文字を n 個並べたストリングを返します。

構文：

── STRING$（n, ┬ x$ ┬ ）──
　　　　　　　　└ m ┘

説明：

n は，並べる文字の数を 1 バイト・コード文字の場合には 0 ～ 255，2 バイト・コード文字の場合には 0 ～ 127 の範囲で指定します。

m は，文字の連続コードを指定します。連続コードは，☞ 解説編 付録 D「コード体系」。

x$ は，並べる文字のストリング式を指定します。

例：

```
10 X$=STRING$(10,45)
20 PRINT X$ "MONTHLY REPORT" X$
RUN
----------MONTHLY REPORT----------
Ok
```

このプログラムは，ハイフンのストリングを連続して表示するために，連続コード 45 を使用しています。

```
10 X$="ABCD"
20 Y$=STRING$(10,X$)
30 PRINT Y$
RUN
AAAAAAAAAA
Ok
```

このプログラムは，与えたストリング ABCD の最初の文字を連続して表示しています。

SWAP ステートメント

目的：　2つの変数の値を交換します。

構文：　SWAP — **変数1, 変数2** —■

説明：　**変数1, 変数2**は，交換する変数または配列要素を指定します。

どんな型の変数（整数，単精度数，倍精度数，ストリング）でも変換できますが，2つの変数は同じ型でなければなりません。

2つの型が違うと Type mismatch（型が合わない）というエラーになります。

例：

```
10 A$="ONE":B$="ALL":C$=" FOR "
20 PRINT A$ C$ B$
30 SWAP A$,B$
40 PRINT A$ C$ B$
Ok
RUN
ONE FOR ALL
ALL FOR ONE
Ok
```

行30の実行後，A$は" ALL "という値になり，B$は" ONE "という値になります。

SYSTEM コマンド

目的：

BASIC から DOS に戻ります。

構文：

SYSTEM ——■

説明：

SYSTEM は DOS に戻る前にすべてのファイルをクローズします。記憶域にあるプログラムや変数は，すべて消去されます。

DOS からバッチ・ファイルを経て BASIC に入り，SYSTEM コマンドで戻ると，バッチ・ファイルの実行は，BASIC に入った点から続けられます。

TAB 関数

T

目的：

行の先頭から（n-1）桁目まで，表示する文字のない桁には，スペースを置きます。

構文：

—— TAB（**n**）——

説明：

n は，文字を表示する（または書き込む）位置を，1～32767 の範囲で指定します。

n が装置の定義された幅より大きい場合，値は n　MOD　幅となります。現在の表示位置がすでに n 桁目を超えている場合は，TAB を行うと次の行の位置 n に進みます。第 1 桁は左端の位置で，右端の位置は WIDTH により定義されます。

TAB は PRINT，PRINT USING，LPRINT，LPRINT USING，PRINT#ステートメント中でのみ使用できます。

TAB 関数で指定したスペースがデータの最後にあると，スペースの後にセミコロンを指定したのと同じように復帰/改行は付け加えられません。

例：

次のプログラムは TAB で項目の桁をそろえて表示しています。

```
10 PRINT "【会社名】" TAB(30) "【電話番号】"
20 READ A$,B$
30 PRINT A$ TAB(30) B$
40 DATA "日本アイ・ビー・エム","03-586-1111"
RUN
【会社名】                    【電話番号】
日本アイ・ビー・エム          03-586-1111
Ok
```

TAN 関数

目的： 角度の正接を返します。

構文： —— TAN (x) ——

説明： x はラジアンで表した角度を数式で指定します。度をラジアンに変換するためには，3.141593/180 を掛けます。

この関数は，単精度で計算した値を返します。倍精度で計算するためには BASIC を始動するときに選択肢/D を指定します。

例： 次のプログラムは，度をラジアンに変換した後，45 度の正接を計算しています。

```
10 PI=3.141593
20 DEGREES=45
30 PRINT TAN(DEGREES*PI/180)
RUN
 1
Ok
```

TIME$ 変数およびステートメント

目的：　現在の時刻を設定あるいは読み取ります。

構文：

— TIME$ —

TIME$＝ — x$ —■

説明：

変数として

現在の文字が8文字ストリングとして返されます。ストリングは，hh:mm:ss形式で，hhは時（00から23），mmは分（00から59）ssは秒（00から59）です。時刻は，DOSの始動時に設定することができます。

ステートメントとして

現在の時刻をx$にhh:mm:ss形式で，設定します。

hh	hhには，時間を0～23の範囲で指定します。これは省略できません。
mm	mmには分を0～59の範囲で指定します。省略すると00が設定されます。
ss	ssは指定しても常に00が設定されます。

TIME$は，time$＝"0:30"と指定することができますが，time$＝":30"とは入力できません。また，time$＝"5:30:25"と指定しても，時刻は530と設定されます。

範囲外の値を指定すると，Illegal function call（違法関数呼び出し）エラーになります。ssに指定した値は無視されますが，ssに0～59以外の値を指定するとIllegal function callエラーになります。また，x$がストリングでないと，Type mismatch（型が合わない）エラーになります。

例：　　次のプログラムは，時刻を連続的に，画面の中央に表示します。

```
10 CLS
20 LOCATE 12,36
30 PRINT TIME$
40 GOTO 20
Ok
```

TIMER 関数

T

目的：　現在の時刻を，単精度の秒単位で返します。

構文：　── TIMER ──

説明：　TIMER 関数で返る値は現在の時刻を秒（積算した秒数）で表したものです。

時刻は DOS の始動時または TIME$ ステートメントで変更することができます。TIMER は読み取り専用関数です。

```
10 TIME$="23:59"
20 FOR I=1 TO 1000
30 PRINT "TIME$=";TIME$,"TIMER=";TIMER
40 NEXT
RUN

TIME$=23:59:00          TIMER= 86340
TIME$=23:59:00          TIMER= 86340
TIME$=23:59:00          TIMER= 86340
TIME$=23:59:00          TIMER= 86341
TIME$=23:59:01          TIMER= 86341
TIME$=23:59:01          TIMER= 86341
.
.
.
TIME$=23:59:59          TIMER= 86399
TIME$=23:59:59          TIMER= 86399
TIME$=00:00:00          TIMER= 0
TIME$=00:00:00          TIMER= 0
```

TIMER ステートメント

目的：　経過時間（秒）ごとの割り込み処理を，可能，禁止，保留状態にします。

構文：

TIMER — ON —■
TIMER — OFF —■
TIMER — STOP —■

説明：

TIMER ステートメントは，ON TIMER (n) ステートメントによって，秒単位で指定した時間が経過するたびに起こる割り込み処理を可能，禁止，保留にします。

TIMER ON は，経過時間ごとの割り込み処理を可能にします。ON TIMER (n) による割り込みを可能にするためには，TIMER ON を実行する必要があります。

TIMER OFF は，経過時間ごとの割り込み処理を禁止にします。TIMER OFF を実行すると時間経過ごとの割り込み処理は実行されません。

TIMER STOP は，経過時間ごとの割り込み処理を保留にします。ただし，割り込み処理をのための経過時間の情報が記憶されるため，TIMER ON を実行すると直ちに割り込み処理が行われます。

例：

```
10 LOCATE 5,2,0:CLS:PRINT "開始!"
20 ON TIMER(5) GOSUB 50
25 TIMER ON:S=TIMER
30 LOCATE 3,2:PRINT "現在の時刻は";TIME$
40 GOTO 30
50 LOCATE 5,2:PRINT TIMER-S;"秒経過しました..."
60 RETURN
Ok
```

TRON および TROFF コマンド

T

目的：　プログラムの実行状態をトレースして調べます。

構文：

TRON ──■

TROFF ──■

説明：　TRON/TROFF は，デバッグ支援のためのコマンドです。このコマンドは間接モードでも使用できます。

TRON はトレース開始の指定です。TRON が指定されると角かっこに囲まれた行番号が表示されます。

TROFF はトレース終了の指定です。

例：

この例は，ループの実行をトレースするために TRON と TROFF を使用しています。角かっこの中の数字は行番号です。角かっこに入っていない数字は，プログラムで表示している J，K，L の値です。

```
10 K=10
20 FOR J=1 TO 2
30 L=K+10
40 PRINT J;K;L
50 K=K+10
60 NEXT
70 END
TRON
Ok
RUN
[10] [20] [30] [40] 1  10  20
[50] [60] [30] [40] 2  20  30
[50] [60] [70]
Ok
TROFF
Ok
```

UNLOCK ステートメント

目的：

オープンされているファイルに対して適用されているアクセス禁止を解除します。

構文：

説明：

n はオープンしたファイルの番号です。値の範囲は，1 ～ 255 です。

番号は，アクセス禁止を解除する範囲をレコード番号で指定します。値の範囲は 1 ～ 4，294,967,295 です。

UNLOCK ステートメントは，LOCK ステートメントで設定されたアクセス禁止を解除します。

LOCK ステートメントや UNLOCK ステートメントで作成したプログラムを実行する前に，共通モジュールが記憶域中に存在している必要があります。このモジュールは，DOS プロンプトの下で SHARE コマンドを実行することによって，記憶域に配置します。

ファイルがランダム・モードでオープンされていて，レコード番号がこのステートメントによって指定されると，その範囲のレコードのアクセス禁止の指定は，解除されます。このとき指定する範囲は，LOCK ステートメントで指定した範囲と同じでなければなりません。違うレコード番号を指定すると Permission denied（書き込み要求拒否）エラーとなります。

ファイルをクローズする前，またはプログラムを終了する前に，すべてのアクセス禁止の指定を解除しておかないと，予期せぬ実行結果となる場合があります。

例：

次の UNLOCK は，正しい指定です。

```
LOCK #1,1 TO 4
LOCK #1,5 TO 8
UNLOCK #1,1 TO 4
UNLOCK #1,5 TO 8
```

次は誤りの例です。

```
LOCK #1,1 TO 4
LOCK #1,5 TO 8
UNLOCK #1,1 TO 8
```

USR 関数

目的：

機械語サブルーチンを，引数を付けて呼び出します。

構文：

```
── USR ─┬─ n ─┬─ (引数) ──
        └─ [1] ┘
```

[1] 0

説明：

n は，DEF USR で機械語サブルーチンに指定した 0 〜 9 の番号を指定します。（☞「DEF USR ステートメント」）。

n の省略値は 0 です。

引数は数式かストリング変数で指定します。

機械語サブルーチンは，CALL ステートメントで呼び出すこともできます。機械語サブルーチンの使用に関しては，☞ 解説編 付録 B「機械語サブルーチン」

VAL 関数

目的：　ストリング x$ の数値を返します。

構文：　—— VAL(x$) ——

説明：　x$ はストリング式です。

例：

VAL 関数は，引数のストリング中のスペース，タブ，および改行文字を無視します。

VAL ("　　-3 ")

は-3 を返します。

x$ の最初の文字が数字でない場合，または，+，−，&，&O，&H，&J，&K のいずれでもない場合，VAL (x$) は 0 (ゼロ) を返します。この場合数字は，連続コード 48 〜 57 (半角)，または 459 〜 468 (全角) に対応する文字です。

数値からストリングへの変換については，STR$ 関数を参照してください。

例：

```
10 '--- VAL 関数の特殊な用法 ---
20 '
30 INPUT "16進数を入力してください。 ===> ",H$
40 DEC = VAL("&H"+H$)
50 PRINT : PRINT"16進数 ";H$;" の10進の値は";DEC;"です。"
60 PRINT : INPUT"8進数を入力してください。===> ",O$
70 DEC = VAL("&O"+O$)
80 PRINT : PRINT"8進数" ;O$; "の10進の値は";DEC;"です。"
90 PRINT : PRINT : INPUT "JIS区点コードを入力してください。===>
",K$
100 CODE=VAL("&K"+K$):KANJI$=CHR$(CODE)
110 PRINT:PRINT"JIS区点コード ";K$;" の連続コードは";CODE;"であ
り，対応する文字は【";KANJI$;"】です。
120 PRINT:PRINT
130 INPUT "JIS16進コードを入力してください。===> ",J$
140 CODE=VAL("&J"+J$):KANJI$=CHR$(CODE)
150 PRINT:PRINT"JIS16進コード ";J$;" の連続コードは";CODE;"であ
り，対応する文字は【";KANJI$;"】です。
Ok
RUN

16進数を入力してください。 ===> 100

16進数 100 の10進の値は 256 です。

8進数を入力してください。===> 100

8進数100の10進の値は 64 です。

JIS区点コードを入力してください。===> 2033

JIS区点コード 2033 の連続コードは 2074 であり，対応する文字は【漢】
です。

JIS16進コードを入力してください。===> 3B7A
```

JIS16進コード 3B7A の連続コードは 2789 であり、対応する文字は【字】です。

V

VARPTR 関数

目的： 変数またファイル制御ブロックのアドレスを返します。

構文：

```
── VARPTR ─ ( ─┬── 変数 ──────┬─ ) ──
               └─ #ファイル番号 ─┘
```

説明： **変数**はアドレスを調べるプログラム中の変数です。VARPTR を呼ぶ前に変数に値が割り当てられないと Illegal function call（違法関数呼出し）エラーになります。

ファイル番号はファイルをオープンしたときに指定した番号です。変数もファイル制御ブロックもアドレスは BASIC のデータ・セグメント内のオフセットです。

返されるアドレスは，0～HFFFF（65535）の範囲です。ただし，整数として得られる値は-32768～32767 ですから，VARPTR で得られた値に加減算を行う場合は注意してください。アドレスは，DEF SEG ステートメントに影響されません。

VARPTR（変数）は，変数で識別されるデータの第 1 バイトのアドレスを返します。このデータの形式は，☞ 解説書 付録 E の「変数の記憶方法」

VARPTR は通常，変数や配列のアドレスを調べてユーザー機械語サブルーチンに渡すために使用します。配列の先頭アドレスを調べるときは VARPTR（A（0））という形（A は配列名）で指定します。

VARPTR（#ファイル番号）は，指定されたファイル制御ブロックの開始アドレスを返します。これは DOS ファイル制御ブロックとは異なります。ファイル制御ブロック形式に関する詳細は，☞ 解説編 付録 E の「BASIC ファイル制御ブロック」。

例：

次の例はランダム・ファイル・バッファー内の先頭アドレスの 1 バイトを読み取ります（この例は変数 FCBADR が正であることを仮定しています)。

```
10 OPEN "DATA.FIL" AS#1
20 GET #1
30 '--- ファイル制御ブロックのアドレスを読む
40 FCBADR=VARPTR(#1)
50 '--- バッファー・アドレスをつくる
60 DATADR=FCBADR+188
70 '--- バッファーの最初のアドレスを読む
80 A%=PEEK(DATADR)
```

次のプログラムは変数からデータを得る例です。行 30 は P のアドレスを調べます。整数データは 2 バイトで，下位バイトが先に記憶されています。行 40 は，P に記憶されている値を PEEK で読み取っています。2 番目のバイトは高位ビットが入っているので 256 をかけます。

```
10 DEFINT A-Z
20 DATA1=500
30 P=VARPTR(DATA1)
40 V=PEEK(P)+256*PEEK(P+1)
50 PRINT V
```

V

VARPTR$ 関数

目的：　変数のアドレスをストリングで返します。

構文：　—— VARPTR$（変数）——

説明：　**変数**はアドレスを調べるプログラム中の変数を指定します。

VARPTR$ は次の 3 バイト・ストリングを返します。

バイト 0	バイト 1	バイト 2
型	変数のアドレスの低位バイト	変数のアドレスの高位バイト

型は変数型を示します。

2 整数
3 ストリング
4 単精度数
8 倍精度数

返される値は次の式で求められる値と同じです。

CHR$（型）＋ MKI$（VARPTR 変数））

PLAY または DRAW のコマンド・ストリング中で変数名を指示するときに，VARPTR$ が使用できます。

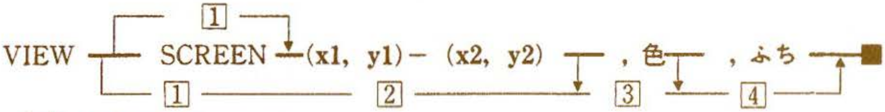
一方法	代替方法
PLAY "XA$;"	PLAY "X"+VARPTR$(A$)
PLAY "O=I;"	PLAY "O="+VARPTR$(I)

VIEW ステートメント

目的：

スクリーン座標系におけるビューポートを定義します。グラフィック・モードでのみ使用できます。

構文：



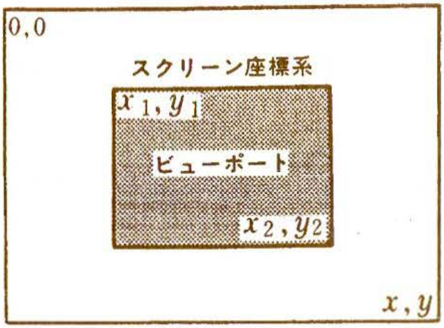
1 相対座標系
2 スクリーン座標系
3 背景色
4 境界線を描かない

説明：

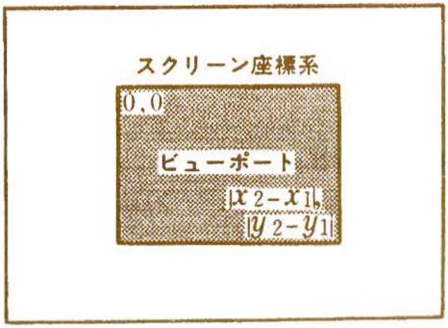
SCREEN は，指定するとビューポートがスクリーン座標系に対して絶対座標に，省略すると相対座標になります。

これを図で表すと次のようになります。SCREEN を指定したときの右下の座標（x，y）は各画面モードの最大座標です。

SCREENを指定したとき



SCREENを省略したとき



(x1, y2) - (x2, y2) は，ビューポートの領域を指定します。（x1, y2）は左上隅で，（x2, y2）は右下隅です。値が画面の座標値を超えると Illegal function call（違法関数呼出し）エラーになります。スクリーン座標系については，SCREEN ステートメントの各画面モードを参照してください。

色はビューポートに塗る色をパレット番号で指定します。

画面モード	パレット番号
単色	（0/1）
4 色カラー	0 〜 3
16 色カラー	0 〜 15

パレット番号の範囲は，画面モードによって異なります。**色**を省略すると，ビューポートは塗りつぶされません。省略時の値はパレット番号 0 です。

ふちは，ビューポートの境界線を描く色をパレット番号で指定します。（ただし，そのスペースがある場合に限ります。）パレット番号の範囲は**色**と同じです。**ふち**を省略すると境界線は描かれません。

ビューポートはスクリーン座標系（各画面モードの座標系）において図形を表示する領域です。ビューポートに関しては，☞ 解説編　第 3 章「ウインドーとビューポート」。

ビューポートの右下の座標値（x2, y2）が，左上の座標値（x1, y1）より小さい場合，双方の値を入れ替えて解釈されますので注意してください。たとえば，

VIEW（100, 100）-（5, 5）は，次のように解釈されます。

VIEW（5, 5）-（100, 100）

x1 と x2 または y2 と y2 を同じ値にはできません。また，ビューポートの最大範囲はスクリーン座標系です。この範囲を超えることはできません。

SCREEN を指定した場合のビューポートは，スクリーン座標系に対して絶対座標になります。

SCREEN を省略した場合のビューポートは，スクリーン座標系に対して相対座標になります。つまり，VIEW を定義した後に描かれる x と y の値には，x1，y2 の座標値が加算されることになります。たとえば，

VIEW (100, 100) - (200, 100)

が実行されてから PSET (0, 0), 3 によって描かれた点は，実際の画面座標において (100,100) の位置になります。

VIEW 定義後，描いた点を実際に目で見ることができるのは，ビューポートとして指定された範囲内の点のみです。たとえば，

VIEW SCREEN (100, 100) - (200, 100)

が実行され，PSET (0, 0), 3 により描かれた点は，画面上で判別できません。それは (0,0) がビューポートの外側にあるためです。

PSET (100, 100), 3 は，ビューポートの内部ですから左上隅の方に点が置かれます。このように指定範囲内の座標値のみを有効とする機能を**クリッピング**といいます。

VIEW で座標値が指定されないときはスクリーン座標系（各画面モードの座標系）がビューポートになります。たとえば，

VIEW (0, 0) - (719, 511)

と指定することと同じ意味になります。

また，画面に複数のビューポートを指定することもできますが，一度に有効なビューポートは 1 つだけです。

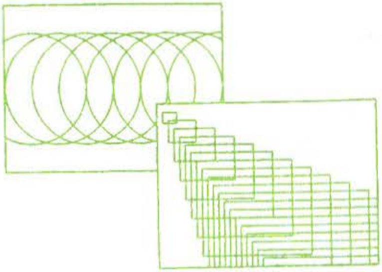
VIEW ステートメントを定義し，図形を描いた後に RUN を実行したり，

SCREEN ステートメントを再び実行すると以前の VIEW ステートメントは無効になります。

例：

次のプログラムはビューポートを相対座標系で使用します。また，範囲外の座標値は無視されることも表現しています。

```
100 '--- ビューポートの相対座標系とクリッピング
110 SCREEN 2,0:VIEW:CLS:KEY OFF
120 VIEW (20,20)-(420,320),,1:GOSUB 150
130 VIEW (300,200)-(700,500),,1:CLS:GOSUB 180
140 END
150 FOR I=1 TO 8
160 CIRCLE (50*I,150),100
170 NEXT:RETURN
180 FOR I=1 TO 20
190 LINE (10*I,15*I)-(35*I,35*I),,B
200 NEXT:RETURN
```



VIEW PRINT ステートメント

目的：　文字の行方向表示範囲を設定します。

構文：　VIEW PRINT [行1 TO 行2]

[1] 行1～行24の範囲

説明：　行1は，表示範囲の最初の行番号を1～24の範囲で指定します。
行2は，表示範囲の最後の行番号を1～24の範囲で指定します。

注）24行めにPFキーの表示があるとき（KEY ONの状態），行1と行2には，1～23の行番号を指定します。行番号24を指定するとIllegal function call（違法関数呼び出し）エラーになります。

行1に行2より大きな値を指定するとIllegal function call（違法関数呼出し）エラーになります。

BASICを操作するときにステートメントや関数，CLS，LOCATE，PRINT，SCREENも，この指定した範囲でのみ有効となります。

画面編集プログラムは，VIEW PRINTステートメントで指定された範囲でスクロールやカーソル移動を行います。

WAIT ステートメント

目的：　装置の入力ポートの状態をモニターする間，プログラム実行を保留します。

構文：

WAIT ─ ポート，n ─┬─，m─┬─■
　　　　　　　　　└─[1]─┘

[1] 0

説明：　ポートには，0 ～ 65535 のポート番号を指定します。

n と m は入力ポート上のビットを調べるために使用します。n は調べる位置を，m は調べるビットのパターンを指定します。n と m の値の範囲は 0 ～ 255 です。m で指定するビットパターンは，調べるビットが 0 なら 1 であり 1 であれば 0 を指定します。WAIT ステートメントは指定された装置入力ポートが特定のビットパターンを形成するまで実行を保留します。ポートで読み取ったデータは整数式 m と XOR しその後 n と AND します。結果が 0 ならループして戻り再びポートからデータを読み取ります。結果が 0 でなければ次のステートメントから実行が続けられます。m を省略すると 0 とみなされます。

また，WAIT ステートメントは，入力ポート上のビット位置を調べることもできます。調べるビットは複数個指定できます。また，そのビットは 0 と 1 のどちらでも調べられます。調べるビットの位置は n の中の対応する位置に 1 を置くことにより指定します。m を省略した場合，0 とみなされるので XOR により入力ポート上の各ビットが 1 か否か調べます。m を指定した場合，n で指定したビット位置に対して 0 か否か調べます。

実行中，WAIT ステートメントは，n の中の 1 によって指定された入力ビットに対しテストしながらループします。これらのビットのうちどれか 1 つが 1 であるとき（あるいは m の中の対応したビットが 1 ならば，0 であるとき）プログラムは次のステートメントから実行されます。このように，WAIT はビット・パターン全体が現れるまで待つことなく，そのうちどれか 1 つでも 1 であれば待つのをやめます。

注）WAIT ステートメントにより無限ループに入ることがあります。ループからぬけでるには，Break キー機能を使用するかシステム・リセットをしてください。

W

WHILE および WEND ステートメント

目的：

与えられた条件が真である間，ループ中の一連のステートメントを実行します。

構文：

WHILE ── 式 ──■
WEND ──■

説明：

式は条件となる数式を指定します。

WHILE は，式が真（ゼロでない）の間，WEND ステートメントまでのループの中のステートメントを繰り返し実行します。式が真でなければ WEND ステートメントの次のステートメントから実行が開始されます。

WHILE...WEND ループはいくつネストすることもできます。各 WEND は最も新しく実行された WHILE と対応します。対応する WEND がない WHILE ステートメントは，WHILE without WEND（WEND のない WHILE）エラーとなり，対応する WHILE がない WEND ステートメントは WEND without WHILE（WHILE のない WEND）エラーとなります。

例：

この例は，ストリング配列 A$ の要素をアルファベット順に分類します。A$ は J 個の要素を持ちます。

```
100 '--- 配列A$のソート ---
110 DIM A$(100)
120 INPUT "配列の個数は";J : IF J>100 THEN 120
130 FOR I=1 TO J
140 PRINT "A$(";I;")";:INPUT A$(I)
150 NEXT I
160 FLIPS=1
170 WHILE FLIPS
180 FLIPS=0
190 FOR I=1 TO J-1
```

```
200 IF A$(I)>A$(I+1) THEN SWAP A$(I),A$(I+1):FLIPS=1
210 NEXT I
220 WEND
230 PRINT "--- 結果 ---"
240 FOR I=1 TO J
250 PRINT "A$(";I;")  ";A$(I)
260 NEXT I
Ok
RUN
配列の個数は? 5
A$( 1 )? J
A$( 2 )? S
A$( 3 )? R
A$( 4 )? A
A$( 5 )? P
--- 結果 ---
A$( 1 )  A
A$( 2 )  J
A$( 3 )  P
A$( 4 )  R
A$( 5 )  S
Ok
```

W

WIDTH ステートメント

目的：

装置やファイルへ文字を出力するときの幅を設定します。

構文：

1 2 前の値

WIDTH — 装置，幅 —■

WIDTH — ファイル番号，幅 —■

説明：

WIDTH **桁，行**

画面の表示範囲を設定します。**桁**と**行**の範囲は次のとおりです。

モード	桁×行
文字モード	80 × 25
4 色カラー・グラフィック・モード	40 × 25
単色/16 色カラー・グラフィック・モード	78 × 25

BASIC 始動時は，80 桁，24 行です。

注） WIDTH で指定する表示範囲が極端に狭い場合，Illegal function Call（違法関数呼び出し）エラーになります。

WIDTH **装置，幅**

装置には，SCRN:，LPT1:，COM1:，COM2: のいずれかを指定します。

幅は，装置に出力する行の幅を 0 ～ 255 の範囲で指定します。

この構文の WIDTH は，装置の新しい出力幅を記憶します。この WIDTH が実行された後，オープンされた装置は，新しい幅で出力されます。この WIDTH が実行される以前にオープンされた装置に対しては無効です。

LPRINT，LLIST，LIST，LPT1: に対しても，この構文は有効です。

WIDTH **ファイル番号，幅**

W

ファイル番号は，SCRN:，LPT1:，COM1:，COM2: のいずれかをオープンしたファイル番号を 0 ～ 255 の範囲で指定します。

幅は，装置に出力する行の幅を 0 ～ 255 の範囲で指定します。

ファイル番号で指定された装置への出力幅が，この構文が実行されるとすぐに，新しい幅に変更されます。これにより，ファイルがすでにオープンされていても幅を変えられます。

パラメーターに範囲外の値が入力されると，Illegal function call（違法関数呼び出し）エラーになります。

プリンターの幅は，省略時 80 に設定されています。この最大幅は DOS の MODE コマンドで変更することができます。詳細は，☞ DOS ユーザーズ・ガイド参照編。

幅に 255 を指定すると，出力行の幅は無限になります。通信ファイルの出力行の幅は 255 です。通信ファイルの幅を変更しても，受信または送信バッファーは変わりません。BASIC は幅で指定された文字ごとに復帰文字を挿入します。

例：

```
10 WIDTH "LPT1:",75
20 OPEN "LPT1:"FOR OUTPUT AS #1
.
.
.
7000 WIDTH #1,40
```

この例は，プリンターの 1 行を 75 文字の幅に設定します。行 20 でオープンしたファイルは，1 行 75 文字になります。行 7000 は，現在オープンされているファイルに対して 1 行を 40 文字に変更します。

WINDOW ステートメント

目的：

ワールド座標系におけるウインドーを定義します。グラフィック・モード のみ使用できます。

構文：

```
WINDOW ─┬─ SCREEN ─┬─ (x1, y1)－(x2, y2) ───■
        └─── 1 ────┘
```

[1] 直交座標系

説明：

SCREEN を指定すると，座標系の X 座標値は左から右に増加し，Y 座標値は上から下へ増加します。SCREEN を省略すると，直交座標系になり，x は左から右へ y は下から上へ増加します。

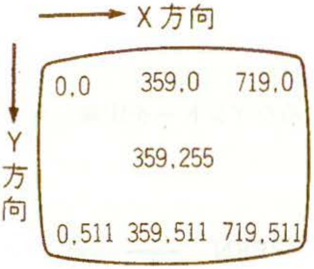
(x1，y1) － (x2，y2) は，ウィンドーの範囲を指定します。
WINDOW で定義するウインドーは，ユーザーが想定するワールド座標系内の使用領域です。したがってスクリーン座標系のように物理的な限界の制約はありません。

BASIC は，このワールド座標系におけるウインドーの座標値を，スクリーン座標系におけるビューポートの座標値に変換します。

たとえば，16 ドット・システムで次の画面モードを設定すると，

```
NEW
SCREEN 2, 0
```

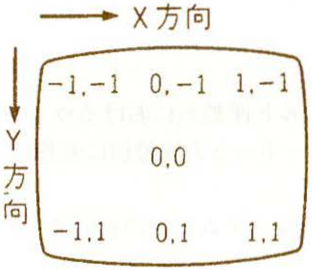
画面の座標系は，次のようになります。



SCREENが指定された場合，座標系のxy方向は，スクリーン座標系と同じです。つまり，このときY座標値は，上から下へ増加します。座標系は，(x1，y1)が左上隅の座標値で，(x2，y2)は，右下隅の座標値です。たとえば，

WINDOW SCREEN (-1，1) - (1，1)

は，次のようにビューポートに対応しています。

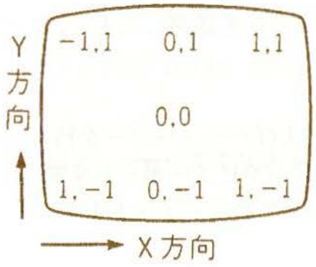


SCREENを省略すると，ウインドーの座標系は直交座標系になります。

たとえば次のように指定します。

WINDOW (-1，1) - (1，1)

ビューポートに対応する座標系は，次のようになります。



すなわち（x1，y1）は，左下隅の座標値であり，（x2，y2）は右上隅の座標値となります。

ウインドーの最初の座標値が，後の座標値より大きい場合は，双方の値が入れ替えて解釈されますので注意が必要です。たとえば，

WINDOW（100，100）−（5，5）

は，次のようになります。

WINDOW（5，5）−（100，100）

また，

WINDOW（− 4，4）−（4，− 4）

は次のようになります。

WINDOW（− 4，− 4）−（4，4）

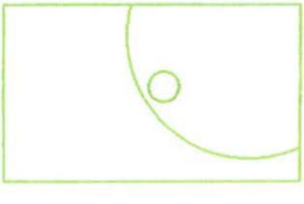
x1 と x2，y1 と y2 は同じ値にできません。

WINDOW ステートメントはクリッピングを行います。クリッピングとは指定した範囲内の座標値のみを有効とするものです。つまりある対象物が範囲内にあるか，範囲外にあるかを認識した上で，範囲外にある点は，無視して表示します。

RUN,SCREEN，およびパラメーターを持たない WINDOW ステートメントの実行は，ウインドーやビューポートを無効にします。画面は，最初の状態に戻ります。したがってビューポートとスクリーン座標系は等しくなります。

例：　これは WINDOW のクリッピング機能と拡大，縮小の例です。

```
100 '--- WINDOWのクリッピングと
110 '            図形の拡大・縮小 ---
120 SCREEN 2,0:CLS
130 '--- ビューポートの設定 ---
140 VIEW(30,20)-(300,180),,1
150 '--- 狭い範囲のウインドーを定義 ---
160 '    (図形は大きくなります)
170 WINDOW(-6,-6)-(6,6)
180 CIRCLE(4,4),5,1
190 '--- 広い範囲のウインドーを定義 ---
200 '  (図形は小さくなります)
210 WINDOW(-50,-50)-(50,50)
220 CIRCLE(4,4),5,1
230 END
Ok
```

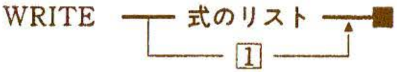


WRITE ステートメント

目的：

データを画面に出力します。

構文：



[1] 復帰/改行を行う

説明：

式のリストは表示する数式やストリング式，またはその両方のリストを指定します。各式は，カンマまたはセミコロンで区切ります。

式のリストを省略すると，復帰/改行のみが行われます。

数値の出力の場合，各項目はカンマで区切られます。ストリングは引用符により区切られます。BASIC はリスト中の最後の項目が表示された後，復帰/改行文字を追加します。

WRITE と PRINT との相違点は，PRINT が区切り文字を指定する必要があるのに対し WRITE は自動的に各項目間にカンマを挿入し，ストリングを引用符で区切る点です。また正の値の前にスペースは置かれません。

例：

この例は WRITE が表示する数値およびストリング値を表しています。

```
10 A=80:B=90:C$="THAT'S ALL"
20 WRITE A,B,C$
RUN
80,90,"THAT'S ALL"
Ok
```

WRITE#ステートメント

目的：　データを順次ファイルに書き込みます。

構文：　**WRITE — #ファイル番号，式のリスト ——■**

説明：　**ファイル番号**は，出力用にオープンしたファイルの番号を指定します。

式のリストはファイルに書き込むデータ（数値，ストリング式）を指定します。WRITE#と PRINT#との違いは，PRINT#が自分で区切り文字を指定するのに対し，WRITE#は自動的に各項目間にはカンマが挿入され，ストリングは引用符で区切られることです。したがってユーザーが，リスト中に区切り文字を指定する必要がありません。また WRITE# は，正の数の前にスペースを置きません。リストの最後の項目が書き込まれた後は，復帰/改行文字が挿入されます。

例：

```
AS="CAMERA"
B$="93604-1"
WRITE#1,A$,B$
```

は，ファイルに次のようなイメージを書き込みます。

```
"CAMERA","93604-1"
```

このファイルを読み取るため次のような INPUT# ステートメントを実行すると，

```
INPUT #1,A$,B$
```

" CAMERA " は A$ に，" 93604-1 " は B$ に入力されます。

バージョン・アップの追加機能一覧

BASIC バージョン K1.1 をもとに,バージョン K2.0,K3.2 で新たに加えられた機能を比較しています。表中の新規または追加に記されているのは,各バージョン番号（K2.0,K3.2）です。

コマンド / ステートメント	新規	追加	追加機能の内容
ABS			
ASC			
ATN			
AUTO			
BEEP			
BLOAD		K2.0	ファイル指定にパス名を指定できる
BSAVE		K2.0	ファイル指定にパス名を指定できる
CALL			
CDBL			
CDBL$			
CHAIN		K2.0	ファイル指定にパス名を指定できる
CHDIR	K2.0		ディレクトリーの変更
CHR$			
CINT			
CIRCLE		K2.0	16 色カラー・グラフィック・モード
CLEAR			
CLOSE			
CLS		K2.0	ビューポート，文字表示範囲など範囲を指定した消去
COLOR		K2.0	16 色カラー・グラフィック・モード
COM（n）			
COMMON			
CONT			
COS			
CSNG			
CSNG$			

コマンド / ステートメント	新規	追加	追加機能の内容
CSRLIN			
CVI,~			
DATA			
DATE$			
DEF FN			
DEF SEG			
DEF USR			
DEF 型			
DELETE		K2.0	指定した行番号以降の削除
DIM			
DRAW		K2.0	16 色カラー・グラフィック・モード
EDIT			
END			
ENVIRON	K2.0		DOS の環境パラメーターを設定する
ENVIRON$	K2.0		DOS の環境パラメーターを読み取る
EOF			
ERASE			
ERDEV,ERDEV$	K2.0	K3.2	エラー装置名とエラー情報を返す (K2.0) エラー情報が拡張された (K3.2)
ERR,ERL			
ERROR			
EXP			
EXTERR	K3.2		DOS 拡張エラー情報を返す
FIELD			
FILES			
FIX			
FOR			
FRE			
GET (ファイル)		K2.0 K3.2	レコード番号が 1 ～ 16777215 に (K2.0) レコード番号が 1 ～ 4294967295 に (K3.2)
GET (グラフィック)		K2.0	16 色カラー・グラフィック・モード
GOSUB			
GOTO			
HEX$			

コマンド / ステートメント	新規	追加	追加機能の内容
IF			
INKEY$			
INP			
INPUT			
INPUT#			
INPUT¥			
INSTR			
INT			
IOCTL	K2.0		独自の装置駆動ルーチンに制御データを渡す
IOCTL$	K2.0		渡した制御データの実行の確認
JIS$			
KEY		K2.0	任意のキー操作に割り込み発生機能を割り当てる（K2.0）
		K3.2	PF11 と PF12 / キー割り込み番号の拡張（K3.2）
KEY (n)		K2.0	任意のキー操作の割り込みを制御（K2.0）
		K3.2	PF11 と PF12 / キー割り込み番号の拡張（K3.2）
			全割り込み定義キーに対する制御（K3.2）
KILL		K2.0	ファイル指定にパス名を指定できる
KLEN			
KPOS			
KTN$			
LEFT$			
LEN			
LET			
LINE		K2.0	16 色カラー・グラフィック・モード，線種の指定（K2.0）
LINE INPUT			
LINE INPUT#			
LIST			
LLIST			
LOAD		K2.0	ファイル指定にパス名を指定できる
LOC			
LOCATE			
LOF			
LOG			
LOCK	K3.2		レコードをアクセス禁止にする

コマンド / ステートメント	新規	追加	追加機能の内容
LPOS			
LPRINT			
LSET,RSET			
MERGE		K2.0	ファイル指定にパス名を指定できる
MID$			
MKDIR		K2.0	新規のディレクトリーの作成
MKI$,~			
NAME		K2.0	ファイル指定にパス名を指定できる
NEW			
NEXT			
OCT$			
ON COM (n)			
ON ERROR			
ON GOSUB			
ON KEY (n)		K2.0	任意のキー操作の割り込みルーチン指定 (K2.0)
		K3.2	PF11 と PF12 / キー割り込み番号の拡張 (K3.2)
ON PLAY (n)	K2.0		背景音楽継続の割り込みルーチン指定
ON TIMER (n)	K2.0		指定時間経過後の割り込みルーチン指定
OPEN		K2.0	ファイル指定にパス名を指定できる (K2.0)
		K3.2	共用ファイルに対するアクセスの制御 (K3.2)
OPEN "COM		K2.2	パリティー検査を行う指定
OPTION BASE			
OUT			
PAINT		K2.0	16 色カラー・グラフィック・モード，色模様の指定
PALETTE		K2.0	16 色カラー・グラフィック・モード
PEEK			
PLAY		K2.0	音符を 1 オクターブ上げるまたは下げる指定
PMAP	K2.0		ウインドーとビューポートの座標交換
POINT		K2.0	ワールド座標系またはスクリーン座標系の座標を返す
POKE			
POS			
PRESET		K2.0	16 色カラー・グラフィック・モード
PRINT			
PRINT#			

コマンド / ステートメント	新規	追加	追加機能の内容
PSET		K2.0	16色カラー・グラフィック・モード
PUT（ファイル）		K2.0	レコード番号が1～16777215に（K2.0）
		K3.2	レコード番号が1～4294967295に（K3.2）
PUT（グラフィック）		K2.0	16色カラー・グラフィック・モード
RANDOMIZE		K2.0	異なる乱数シードを得るためのTIMER関数の使用
READ			
REM			
RENUM			
RESET			
RESTORE			
RESUME			
RETURN			
RIGHT$			
RMDIR	K2.0		ディレクトリーの削除
RND			
RSET			
RUN		K2.0	ファイル指定にパス名を指定できる
SAVE		K2.0	ファイル指定にパス名を指定できる
SCREEN（F）			
SCREEN（S）		K2.0	16色カラー・グラフィック・モード画面の設定
SGN			
SHELL	K2.0		DOSのコマンドを実行する
SIN			
SOUND			
SRACE$			
SPC			
SQR			
STOP			
STR$			
STRING$			
SWAP			
SYSTEM			
TAB			
TAN			

コマンド / ステートメント	新規	追加	追加機能の内容
TIME$			
TIMER (F)	K2.0		時刻を秒単位で返す
TIMER (S)	K2.0		指定した経過時間ごとの割り込み処理を制御する
TRON,TROFF			
UNLOCK	K3.2		レコードに対するアクセス禁止の指定を解除する
USR			
VAL			
VARPTR			
VARPTR$			
VIEW	K2.0		スクリーン座標系のビューポートを定義する
VIEWPRINT	K2.0		文字の行方向の表示範囲を設定する
WAIT			
WHILE,WEND			
WIDTH			
WINDOW	K2.0		ワールド座標系におけるウインドーを定義する
WRITE			
WRITE#			

＊（F）は関数，（S）はステートメントを表します。

付録A

メッセージ集

メッセージ集

プログラムの実行を中止させるようなエラーを BASIC が探知すると，エラー・メッセージを表示します ON ERROR ステートメントおよび ERR と ERL 変数を使って，BASIC プログラム中のエラーを割り込み処理し，テストすることができます。

この付録では，BASIC のエラー・メッセージを，それが発生したときに ERR にセットされるエラー番号と共にリストにしています。

番号　メッセージ

54 Bad file mode（ファイル・モードの誤り）

順次ファイルに対して，PUT や GET を使おうとした。または，オープンされていないファイルに対して，入出力を行おうとした。または，入力，出力，アペンド，またはランダム以外のファイル・モードで OPEN を実行した。

OPEN ステートメントが正しく入力，実行されるようにしてください。GET および PUT はランダム・ファイルを必要とします。

このエラーはまた，文字形式（内部コード形式）でないファイルをマージしようとしたときにも起こります。この場合，正しいファイルをマージするようにしてください。必要ならば，プログラムを読み込み，再びオプション A を使って SAVE してください。

64 Bad file name（ファイル名の誤り）

BLOAD，BSAVE，KILL，NAME，OPEN，FILES ステートメントのファイル名に，正しくない形式が使われている。

正しいファイル名に関する情報を解説編 第 3 章の「ファイル指定とパス」でチェックし，エラーとなったファイル名を修正してください。

52 Bad file number（ファイル指定の誤り）

ファイル番号の指定が誤っている。

目的ファイルがオープンされていることを確かめ，ステートメントに正しいファイル指定が行われているかどうかチェックしてください。

63 Bad record number（レコード番号の誤り）

PUT あるいは GET ステートメント中で，レコード番号が許された最大値 4294967295 より大きいかゼロに等しい。

正しいレコード番号を使うよう PUT あるいは GET ステートメントを修正してください。

17 Can't continue（継続不可能）

次のようなプログラムを続行するために CONT を使おうとした。

- エラーのために停止されたプログラム。
- 実行の中断の間に修正されたプログラム。
- 存在しないプログラム。

プログラムが記憶域に読み込まれているかどうか確認し，実行には RUN を使ってください。

＊＊Can't continue after SHELL（SHELL ステートメント後の実行ができない）

SHELL ステートメントによるプログラムの実行後，記憶領域が足りず BASIC を続けられない。エラー番号なし。オープンされているすべてのファイルはクローズされ制御は DOS に戻ります。

69 Commutication buffer overflow（通信バッファーのあふれ）

通信入力ステートメントが実行されたが，入力バッファーがすでにいっぱいになっている。

この状態になったとき，入力をやり直すには，ON ERROR ステートメントを使用してください。以降の入力はこの状態を消去しようとします。ただし，プログラムが処理できるよりも速く，文字の受信が続くと，消去はできません。この場合，次のような解決策が可能です。

- BASIC を開始するときに，選択肢/C: を使って通信バッファーのサイズを大きくする。
- 他のコンピューターと「初期接続手順」プロトコルを交わし，こちらが追いつけるよう十分長く送信を停止するよう告げる（「解説編　付録C　通信」の中の例を参照してください。）
- 送受信に，より低い伝送速度（bps）を使う。

25 Device Fault（装置の故障）

インターフェース・アダプターからハードウェア・エラー表示が返された。

このメッセージは，データを通信ファイルに送信しているときにも起こります。この場合，メッセージは，OPEN " COM... ステートメントでテストされた信号が指定された時間内に検出されなかったことを示します。

57 Device I/O Error（装置入出力エラー）

装置入出力操作中にエラーが起こった。DOS はエラーから回復できない。

通信データの受信中の場合，このエラーはオーバーラン，フレーム指示，中断，パリティーの誤りによって起こります。7 以下のデータ・ビットを持つデータを受信した場合，エラーの起ったバイトの 8 番目のビットが on になります。

24 Device Timeout（装置タイムアウト）

BASIC が予定の時間内に入出力装置から情報を受け取らなかった。
通信ファイルにおいては，このメッセージは，OPEN " COM... でテストされた各信号が，指定された時間内に検出されなかったことを示します。
もう一度操作してみてください。

68 Device Unavailable（装置が使用不可能）

存在しない装置に対してファイルをオープンしようとした。

その装置をサポートするハードウェアがない，あるいはその装置を使用禁止にしている（たとえば，BASIC を開始する BASIC コマンド上で /C:0 オプションを使ったとします。これにより，通信装置は使用禁止になります。また，プリンターのスイッチが入っていない場合などにも起こります）。

装置を正しく接続してください。必要ならば，次のコマンドを入力してください。

SYSTEM

これによりユーザーは DOS へ戻り，そこで BASIC コマンドを再入力できます。

66 Direct statement in file（ファイル中に直接ステートメントがある）

文字形式（内部コード形式）ファイルの読み込みまたは連鎖中に，直接ステートメントに出会った。LOAD または CHAIN は中止される。
文字形式ファイルは，行番号が前についたステートメントだけから構成されなくてはなりません。このエラーは，入力ストリーム中の改行文字のせいで起こることもあります。

61 Disk full（ディスクがいっぱいである）

ディスク記憶空間がすべて使用されている。このエラーが起こるとファイルはクローズされる。

ディスク上に不要なファイルがある場合は，それらを消去してください。または，新しいディスケットを使用してください。その後，操作をやり直すかプログラムを再実行してください。

72 Disk Media error（ディスク・メディア・エラー）

アダプター・カードが，ハードウェアまたはメディアの故障を探知した。ふつうこれは，ディスケットが悪くなっていることを意味します。

既存のファイルを新しいディスケットにコピーし，悪くなったディスケットを再びフォーマットしてください。

71 Disk not Ready（ディスクが作動可能でない）

ディスケット・ドライブのレバーが開いている，あるいはディスケットがディスケット・ドライブに入っていない。

ディスケットをディスケット・ドライブに正しく入れ，プログラムを続行してください。

11 Division by zero（ゼロでの除算）

式の中で，ゼロで除算を行おうとした。またはゼロに負の数の累乗を行おうとした。

プログラムは実行を続けますから，この状態を修正する必要はありません。ただし除算の結果は被除数の符号をつけた数値範囲の上限値となり，累乗算の結果は正の上限値となります。

10 Duplicate Definition（二重定義）

同じ配列に対してサイズを 2 回定義しようとした。これは次のような場合が考えられる。

- 2 つの DIM ステートメントで同じ配列を定義した。
- 省略時の 1 次元で添字の最大値 10 の配列が与えられたあと，プログラムはその配列を定義する DIM ステートメントに出会った。

● 配列が DIM ステートメントまたは省略時の値で定義された後，プログラムは矛盾する OPTION BASE ステートメントに出会った。

OPTION BASE ステートメントを必ず配列の使用前に実行されるように移動してください。あるいは各配列がただ一度定義されるようプログラムを修正してください。

50 FIELD overflow (FIELD のあふれ)
ランダム・ファイルのレコード長として OPEN ステートメント内で指定したバイト数より多いバイト数を FIELD ステートメントが割り振ろうとしている。または，ランダム・ファイルへの順次入出力（PRINT #，WRITE#，INPUT#）中に FIELD バッファーの終わりが起こった。

OPEN ステートメントと FIELD ステートメントをチェックして，両者が対応するようにしてください。ランダム・ファイルへの順次入出力中の場合，読み取りまたは書き込まれるデータ長がランダム・ファイルのレコード長を超えないようにしてください。

58 File already exists (ファイルがすでに存在する)
NAME ステートメントで指定されたファイル名が，ディスク上ですでに使用されている。
別な名前を使って NAME コマンドをやり直してください。

55 File already open (ファイルはすでにオープンされている)
オープンしようとしたファイルはすでにオープンされていた。またはオープンしているファイルに対し KILL を行おうとした。

ファイルの OPEN は 1 回だけ実行してください。KILL を使う前にはファイルをクローズしてください。

53 File not found (ファイルが見つからない)
LOAD，KILL，NAME，FILES または OPEN ステートメントで参照しているファイルが，指定したドライブの現行ディレクトリー内に

存在しない。

指定したドライブに正しいディスケットが入っているか，およびファイル指定が正しく入力されたかを検査してください。または，現行ディレクトリーの変更後，操作をもう一度試みてください。

26 FOR without NEXT (NEXT のない FOR)

FOR ステートメントに対応する NEXT がない。

NEXT ステートメントを含むプログラムに修正してください。

12 Illegal direct (不正な直接モード)

直接モードでは使えないステートメント (DEF FN など) を直接モードで入力しようとした。

このステートメントはプログラム行の一部として間接モードで入力しなくてはなりません。

5 Illegal function call (違法関数呼び出し)

有効範囲外のパラメーターがシステム関数に渡された。
このエラーはまた，次のような場合の結果として起こることがあります。

- 負または不当に大きな添字
- 負の数に整数でない数の累乗を行おうとした。
- DEF USR で開始アドレスを定義しないうちに，USR 関数を呼び出した。
- 関数あるいはステートメントの引数が不適切。
- 記憶保護された BASIC プログラムにリストあるいは編集を行おうとした。
- DELETE ステートメントによって存在しない行番号を削除しよ

うとした。

プログラムを修正してください。特定のステートメントまたは関数についての情報は，「BASICの文法」を参照してください。

62 Input past end (入力が終わりを超えた)

これはファイルの終わりエラーです。空白（空の）ファイルに対し入力ステートメントが実行された，あるいは順次ファイル中の全データがすでに入力された後に，入力ステートメントが実行された。
このエラーを回避するには，ファイルの終わりを探知するためにEOF関数を使ってください。

51 Internal error (内部エラー)

BASICで内部エラーが起こった。

オリジナルBASICインタープリターをディスケットからコピーし直してください。ハードウェアをチェックしてもう一度操作してください。

23 Line buffer overflow (行バッファーのあふれ)

文字の多すぎる行を入力しようとした。

行の中の複数ステートメントを，2行以上になるよう分割してください。可能なら定数の代わりにストリング変数を使ってください。

22 Missing operand (オペランドがない)

式の中に，後ろにオペランドのない演算子，*やORなどがある。

式の中に必要なオペランドをすべて含むようにしてください。

1 NEXT without FOR (FORのないNEXT)

NEXTステートメント中の変数の順序とFORステートメントによる変数の順序が異なる場合や，NEXTステートメント中の変数がいくつかのFORステートメントの変数のいずれとも対応しない場合。

NEXTステートメントに対応するFORを持つようプログラムを修正

してください。

19 No RESUME (RESUMEがない)

エラー状態またはERRORステートメントの結果，活動中のエラー割り込み処理ルーチンにプログラムが分岐したが，そのルーチンにRESUMEステートメントがなかった（エラー割り込み処理ルーチン内で，物理的なプログラムの終わりに出会った)。

プログラムの実行を続けるために，エラー割り込み処理ルーチン内に必ずRESUMEを含むようにしてください。またエラー割り込み処理ルーチンにON ERROR GOTO 0ステートメントを付け加えて，エラーの割り込みを解除し，エラー・メッセージを表示させることができます。

4 Out of data (データがない)

READステートメントが，DATAステートメントで与えられた以上のデータを読もうとしている。

プログラム中のすべてのREADステートメントに十分なだけの定数がDATAステートメント内で与えられるように，プログラムを修正してください。

7 Out of memory (記憶域を超えた)

プログラムが大きすぎる。FORループまたはGOSUBが多すぎる，変数が多すぎる，式が複雑すぎる，PAINTステートメントの指示が複雑すぎるなどの理由が考えられる。

スタック空間または記憶域を，より多く取っておくために，プログラム開始時にCLEARを実行しておく方法もあります。

14 Out of string space (ストリング空間が足りない)

BASICは，記憶域がいっぱいになるまでストリング空間を動的に割り振ります。このメッセージは，ストリング変数によりBASICがハウス

クリーニング後残っていた自由記憶域容量を超えたことを示します。

6 Overflow（桁あふれ）

数の絶対値が大きすぎて，BASIC の数値形式は表現できない。整数の桁あふれは実行を停止させます。それ以外の場合は，数値範囲の上限値に適切な符号をつけたものが結果として与えられ，実行は続けられます。

整数の桁あふれを修正するには，より小さな数を使うか，あるいは単精度または倍精度変数に変更してください。

注） 数が小さすぎて BASIC の数値形式では表現できない場合，下位桁あふれ状態となります。これが起こると，結果はゼロとなり，エラーにはならず実行が続けられます。

75 Path/file across error（パス指定の誤り）

MKDIR，CHDIR，RMDIR または他のファイル指定におけるパス名の指定で，DOS が，ファイルへのパスの指定に誤りを見つけた。この操作はやり直す必要があります。

76 Path not found（パスが見つからない）

MKDIR，CHDIR，RMDIR または他のファイル指定におけるパス名の指定で，DOS が，指定されたパスを見つけられなかった。この操作はやり直す必要があります。

70 Permisson denied（書き込み要求拒否）

このエラーは次の状態の時に起こります。

- 書き込み防止されているディスクに対して書き込みを行おうとした。
- ファイルをオープンする際に OPEN ステートメントの指定に誤りがあった。
- すでに OPEN ステートメントまたは LOCK，UNLOCK ステートメントで書き込み防止についての処置がなされているファイルに

対して再び設定しようとした場合。

- 他の処理によってすでに書き込み保護されているファイルに対して書き込みを行おうとした。

- データの書き込み終了後，他の処理がそのファイルを書き込み保護してアクセスしている時に，そのファイルをクローズしようとした。

  他の処理が書き込み保護の解除を行うまでファイル・クローズの命令を繰り返してください。
  このメッセージは，END ステートメントで同様の状態になった場合も起こります。

- LOCK，UNLOCK ステートメントを使用する前に DOS の SHARE コマンドが実行されていなかった。

74 Rename across disks（ディスクを超えて指定した）
ファイル名を新たに変更するときに現行のディスクではなく他のディスクを指定した。この操作は取り消されます。

20 RESUME without error（エラーなしの RESUME）
エラーを割り込み処理していないのにプログラム中に RESUME ステートメントがあった。

エラー処理ルーチンに入るのは，エラーが起ったときまたは ERROR ステートメントが実行されたときのみです。
プログラムがエラー割り込み処理ルーチンの中に落ち込まないようにするには，エラー割り込み処理ルーチンの前に STOP または END ステートメントを置いたほうがよいかもしれません。

3 RETURN without GOSUB（GOSUB のない RETURN）
RETURN ステートメントより前に，対応する GOSUB ステートメントがない。

プログラムを修正してください。プログラムがサブルーチンの中に落ち

込まないようにするには，サブルーチンの前に STOP または END を置いたほうがよいかもしれません。

16 String formula too complex **(ストリング式が複雑すぎる)**
ストリング式が長すぎるか複雑すぎる。
小さい式に分割してください。

15 String too long **(ストリングが長すぎる)**
255 文字より長いストリングを作ろうとした。

小さいストリングに分割してください。

9 Subscript out of range **(添字が範囲外)**
配列要素の添字が配列の次元を超えているか，あるいは間違っている。

配列変数の用法をチェックしてください。配列でない変数に添字をつけたか，組み込み関数を誤ってコーディングしたかもしれません。

2 Syntax error **(構文エラー)**
プログラム行の中に，対になっていないかっこや，綴りの間違ったコマンドやステートメントのような誤った文字列，誤った句読点があるか，DATA ステートメント内のデータが READ ステートメント内の変数型（数値またはストリング）と一致しない。

このエラーが起こると BASIC プログラム編集プログラムが自動的にエラーの起った行を表示しますからその行のプログラムを修正してください。

67 Too many files **(ファイルが多すぎる)**
現行上のディレクトリーがすべていっぱいのとき，ファイル指定が誤っているときに，新しいファイルを作成しようとした（SAVE または OPEN を使って)。

ファイル指定が間違いない場合で，現行ディレクトリーがいっぱいのときは，あいている他のディレクトリーを使用するか，フォーマットされ

た新しいディスケットを使ってやり直してください。

13 Type mismatch (型が合わない)

数値を必要とする場所にストリング値を与えた，またはストリング値のあるべき所に数値を入れた。

このエラーはまた，単精度と倍精度のような違った型の変数を SWAP しようとした場合にも起こります。

8 Undefined line number (行番号が存在しない)

ステートメント，コマンド内で参照している行番号が，プログラム中に存在しない。

プログラム中の行番号をチェックし，正しい行番号を使ってください。

18 Undefined user function (ユーザー関数が未定義)

DEF FN ステートメントでまだ定義されていない関数を呼び出した。

関数を使用する前に，プログラムが DEF FN ステートメントを実行していることを確認してください。

Unprintable error (表示不能エラー)

発生したエラー状態に対して該当するエラー・メッセージがない。これは通常，定義されていないエラー・コードを ERROR ステートメントで使用したときに起こります。

作成したすべてのエラー・コードを処理するようユーザー・プログラムをチェックしてください。

30 WEND without WHILE (WHILE のない WEND)

対応する WHILE が実行されないうちに，WEND が現れた。

各 WEND に WHILE が対応するようプログラムを直してください。

29 WHILE without WEND (WEND のない WHILE)

WHILE ステートメントに対応する WEND がない。すなわち，プログラムの物理的終わりになったとき，WHILE がまだ活動状態だった。

それぞれの WHILE が対応する WEND を持つようプログラムを直してください。

※※You can not SHELL to BASIC (子プログラムとしての BASIC を実行しようとした)

子プログラムとしての BASIC の実行が発見された。エラー番号なし。BASIC のもとで BASIC を呼び出してはいけません。

エラー番号とメッセージ

番号	メッセージ
1	NEXT without FOR
2	Syntax error
3	RETURN without GOSUB
4	Out of data
5	Illegal function call
6	Overflow
7	Out of memory
8	Undefined line number
9	Subscript out of range
10	Duplicate Definition
11	Division by zero
12	Illegal direct
13	Type mismatch
14	Out of string space
15	String too long
16	String formula too complex
17	Can't continue
18	Undefined user function
19	No RESUME
20	RESUME without error
22	Missing operand
23	Line buffer overflow
24	Device Timeout
25	Device Fault
26	FOR without NEXT
29	WHILE without WEND
30	WEND without WHILE
50	FIELD overflow
51	Internal error

番号	メッセージ
52	Bad file number
53	File not found
54	Bad file mode
55	File already open
57	Device IO error
58	File already exists
61	Disk full
62	Input past end
63	Bad record number
64	Bad file name
66	Direct statement in file
67	Too many files
68	Device unavailable
69	Communication buffer overflow
70	Permission denied
71	Disk not ready
72	Disk media error
73	Advanced feature
74	Rename across disks
75	Path/file across error
76	Path not found
--	Unprintable error

付録B

参照資料

変数型宣言文字

変数型	型宣言文字	バイト数
ストリング	$	可変
整数	%	2
単精度数	!	4
倍精度数	#	8

演算子

算術演算子	演算	例
^	べき乗	X ^ Y
－	負の数	－X
*	乗算	X*Y
/	小数点	X / Y
¥	整数除算	X¥Y
MOD	剰余の計算	X MOD Y
＋	加算	X＋Y
－	減算	X－Y

関係演算子	意味	例
=	等しい	X=Y
<>または><	等しくない	X<>Y,X><Y
<	より小さい	X<Y
>	より大きい	X>Y
<= または =<	以下である	X<=Y,X=<Y
>= または =>	以下である	X>=Y,X=>Y

論理演算子			
NOT	X		NOT X
	T		F
	F		T
AND	X	Y	X AND Y
	T	T	T
	T	F	F
	F	T	F
	F	F	F
OR	X	Y	X OR Y
	T	T	T
	T	F	T
	F	T	T
	F	F	F
XOR	X	Y	X XOR Y
	T	T	F
	T	F	T
	F	T	T
	F	F	F
EQV	X	Y	X EQV Y
	T	T	T
	T	F	F
	F	T	F
	F	F	T
IMP	X	X	X IMP Y
	T	T	T
	T	F	F
	F	T	T
	F	F	T

ファイル指定とパス名

ファイル指定

次のどちらかの形式のストリング式で指定。

装置名:

ドライブ指定子:ファイル名.エクステンション

装置名は下記の装置名表のいずれかを指定。**ドライブ指定子**は下記のいずれかを指定。**ファイル名**は 1 〜 8 文字。**エクステンション**は 1 〜 3 文字。

パス名

次の形式のストリング式で指定。

ドライブ指定子:パス名

ドライブ指定子は下記のいずれかを指定。**パス名**はディレクトリーを円記号(¥)で区切って並べたもので 63 文字以内のストリング式で指定。

装置名とドライブ指定子

装置名

KYBD:	キーボード入力のみ。
SCRN:	画面。出力のみ。
LPT1:	第 1 プリンター。順次出力とランダム出力。
COM1:	通信カード 1。
COM2:	通信カード 2。

ドライブ指定子

A:	第 1 ディスケット・ドライブ。入力と出力。
B:	第 2 ディスケット・ドライブ。入力と出力。
C:	第 3 ディスケット・ドライブ。入力と出力。

画面モード

画面モードと色指定

画面	名称	桁数	色番号	パレット番号
0,0	単色文字	80	−	−
0,1	カラー文字	80	0 〜 15	−
1,0	単色グラフ	78	−	(0 / 1)
1,1	4 色/16 色カラーグラフ	40/78	0 〜 15/0 〜 63	0 〜 3/0 〜 15
2,0	単色グラフ	78	−	(0 / 1)
2,1	16 色カラー・グラフ	78	0 〜 63	0 〜 15

グラフィック画面の点の数

画面	16 ドット単色	16 ドット・カラー	24 ドット単色	24 ドット・カラー
単色	720 × 512	720 × 512	1024 × 768	1024 × 768
4 色カラー	-	360 × 512	-	-
16 色カラー	-	720 × 512	-	1024 × 768

画面モードの表は以下の点に留意して参照してください。

- 画面は SCREEN ステートメントで指定する場合の**モード**，**バースト**の形式で示しています。
- 「名称」の「文字」は文字モードを，「グラフ」はグラフィック・モードを表します。
- 単色グラフィック・モードの場合，パレットとしては扱わないため（ ）を付けてあります。ただし，グラフィック・ステートメントではパラメーター**パレット番号**に 0 または 1 を指定します。

- 画面「1, 1」は/（スラッシュ）の左側が 16 ドット・システム，右側が 24 ドット・システムです。

- 単色システムの場合，「名称」のカラー画面はすべて単色画面になります。

- 旧システムの場合のみ，画面「2, 1」は単色グラフィック・モードになります。

- 行数はすべて 25 行表示できますが，25 行目は「入力表示行」としてシステムが使用します。

数学関数の公式

BASICに無い数学関数はDEF FNステートメントで関数定義をしておくと便利です。関数定義を記述する場合は，次の式を参考にしてください。

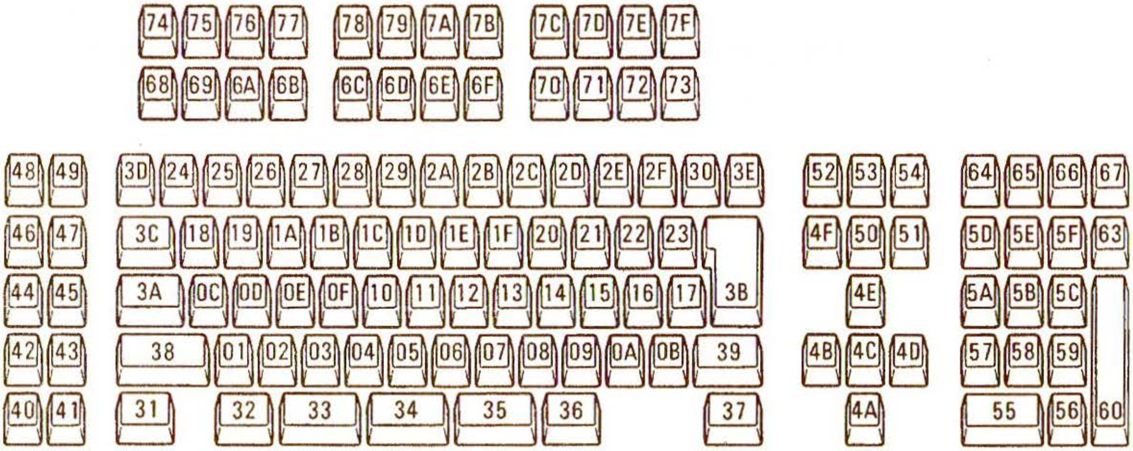
関数	等価式
Bを底とする対数	LOGB(X)=LOG(X)/LOG(B)
SEC関数	SEC(X)=1/COS(X)
COSEC関数	CSC(X)=1/SIN(X)
COT関数	COT(X)=1/TAN(X)
ARCSIN関数	ARCSIN(X)=ATN(X/SQR(1-X*X))
ARCCOS関数	ARCCOS(X)=1.570796-ATN(X/SQR(1-X*X))
ARCSEC関数	ARCSEC(X)=ATN(SQR(X*X-1))+(X<0)*3.141593
ARCCOSEC関数	ARCCSC(X)=ATN(1/SQR(X*X-1))+(X<0)*3.141593
ARCCOT関数	ARCCOT(X)=1.57096-ATN(X)
SINH関数	SINH(X)=(EXP(X)-EXP(-X))/2
COSH関数	COSH(H)=(EXP(X)+EXP(-X))/2
TANH関数	TANH(X)=(EXP(X)-EXP(-X))/(EXP(X)+EXP(-X))
SECH関数	SECH(X)=2/(EXP(X)+EXP(-X))
COSECH関数	CSCH(X)=2/(EXP(X)-EXP(-X))
COTH関数	COTH(X)=(EXP(X)+EXP(-X))/(EXP(X)-EXP(-X))
ARCSINH関数	ARCSINH(X)=LOG(X+SQR(X*X+1))
ARCCOSH関数	ARCCOSH(X)=LOG(X+SQR(X*X-1))
ARCTANH関数	ARCTANH(X)=LOG((1+X)/(1-X))/2
ARCSECH関数	ARCSECH(X)=LOG((1+SQR(1-X*X))/X)
ARCCOSECH関数	ARCCSCH(X)=LOG((1+SGN(X)*SQR(1+X*X))/X)
ARCCOTH関数	ARCCOTH(X)=LOG((X+1)/(X-1))/2

DEF FN ステートメントで関数を定義するには，次のように記述します。

```
DEF FNARCSINH(X)=LOG(X+SQR(X*X+1))
```

実際に計算に使用するところでは，FNARCSINH（Y）の形式で関数を使用します。

キーボードと走査コード



数値変換表

10進	16進	2進	10進	16進	2進	10進	16進	2進	10進	16進	2進
0	00	00000000	32	20	00100000	64	40	01000000	96	60	01100000
1	01	00000001	33	21	00100001	65	41	01000001	97	61	01100001
2	02	00000010	34	22	00100010	66	42	01000010	98	62	01100010
3	03	00000011	35	23	00100011	67	43	01000011	99	63	01100011
4	04	00000100	36	24	00100100	68	44	01000100	100	64	01100100
5	05	00000101	37	25	00100101	69	45	01000101	101	65	01100101
6	06	00000110	38	26	00100110	70	46	01000110	102	66	01100110
7	07	00000111	39	27	00100111	71	47	01000111	103	67	01100111
8	08	00001000	40	28	00101000	72	48	01001000	104	68	01101000
9	09	00001001	41	29	00101001	73	49	01001001	105	69	01101001
10	0A	00001010	42	2A	00101010	74	4A	01001010	106	6A	01101010
11	0B	00001011	43	2B	00101011	75	4B	01001011	107	6B	01101011
12	0C	00001100	44	2C	00101100	76	4C	01001100	108	6C	01101100
13	0D	00001101	45	2D	00101101	77	4D	01001101	109	6D	01101101
14	0E	00001110	46	2E	00101110	78	4E	01001110	110	6E	01101110
15	0F	00001111	47	2F	00101111	79	4F	01001111	111	6F	01101111
16	10	00010000	48	30	00110000	80	50	01010000	112	70	01110000
17	11	00010001	49	31	00110001	81	51	01010001	113	71	01110001
18	12	00010010	50	32	00110010	82	52	01010010	114	72	01110010
19	13	00010011	51	33	00110011	83	53	01010011	115	73	01110011
20	14	00010100	52	34	00110100	84	54	01010100	116	74	01110100
21	15	00010101	53	35	00110101	85	55	01010101	117	75	01110101
22	16	00010110	54	36	00110110	86	56	01010110	118	76	01110110
23	17	00010111	55	37	00110111	87	57	01010111	119	77	01110111
24	18	00011000	56	38	00111000	88	58	01011000	120	78	01111000
25	19	00011001	57	39	00111001	89	59	01011001	121	79	01111001
26	1A	00011010	58	3A	00111010	90	5A	01011010	122	7A	01111010
27	1B	00011011	59	3B	00111011	91	5B	01011011	123	7B	01111011
28	1C	00011100	60	3C	00111100	92	5C	01011100	124	7C	01111100
29	1D	00011101	61	3D	00111101	93	5D	01011101	125	7D	01111101
30	1E	00011110	62	3E	00111110	94	5E	01011110	126	7E	01111110
31	1F	00011111	63	3F	00111111	95	5F	01011111	127	7F	01111111

10進	16進	2進	10進	16進	2進	10進	16進	2進	10進	16進	2進
128	80	10000000	160	A0	10100000	192	C0	11000000	224	E0	11100000
129	81	10000001	161	A1	10100001	193	C1	11000001	225	E1	11100001
130	82	10000010	162	A2	10100010	194	C2	11000010	226	E2	11100010
131	83	10000011	163	A3	10100011	195	C3	11000011	227	E3	11100011
132	84	10000100	164	A4	10100100	196	C4	11000100	228	E4	11100100
133	85	10000101	165	A5	10100101	197	C5	11000101	229	E5	11100101
134	86	10000110	166	A6	10100110	198	C6	11000110	230	E6	11100110
135	87	10000111	167	A7	10100111	199	C7	11000111	231	E7	11100111
136	88	10001000	168	A8	10101000	200	C8	11001000	232	E8	11101000
137	89	10001001	169	A9	10101001	201	C9	11001001	233	E9	11101001
138	8A	10001010	170	AA	10101010	202	CA	11001010	234	EA	11101010
139	8B	10001011	171	AB	10101011	203	CB	11001011	235	EB	11101011
140	8C	10001100	172	AC	10101100	204	CC	11001100	236	EC	11101100
141	8D	10001101	173	AD	10101101	205	CD	11001101	237	ED	11101101
142	8E	10001110	174	AE	10101110	206	CE	11001110	238	EE	11101110
143	8F	10001111	175	AF	10101111	207	CF	11001111	239	EF	11101111
144	90	10010000	176	B0	10110000	208	D0	11010000	240	F0	11110000
145	91	10010001	177	B1	10110001	209	D1	11010001	241	F1	11110001
146	92	10010010	178	B2	10110010	210	D2	11010010	242	F2	11110010
147	93	10010011	179	B3	10110011	211	D3	11010011	243	F3	11110011
148	94	10010100	180	B4	10110100	212	D4	11010100	244	F4	11110100
149	95	10010101	181	B5	10110101	213	D5	11010101	245	F5	11110101
150	96	10010110	182	B6	10110110	214	D6	11010110	246	F6	11110110
151	97	10010111	183	B7	10110111	215	D7	11010111	247	F7	11110111
152	98	10011000	184	B8	10111000	216	D8	11011000	248	F8	11111000
153	99	10011001	185	B9	10111001	217	D9	11011001	249	F9	11111001
154	9A	10011010	186	BA	10111010	218	DA	11011010	250	FA	11111010
155	9B	10011011	187	BB	10111011	219	DB	11011011	251	FB	11111011
156	9C	10011100	188	BC	10111100	220	DC	11011100	252	FC	11111100
157	9D	10011101	189	BD	10111101	221	DD	11011101	253	FD	11111101
158	9E	10011110	190	BE	10111110	222	DE	11011110	254	FE	11111110
159	9F	10011111	191	BF	10111111	223	DF	11011111	255	FF	11111111

目的別索引

あ 行

アクセス禁止
　～を解除する …… 文 292
　～を指定する …… 文 156
時間が経過したときに実行するルーチンを
　　指定する …… 文 183
色
　～を指定する …… 文 32
　～を塗る …… 文 200
印刷制御文字を指定するには
　　…… 解 E-10
ウインドーを設定する …… 文 313
エラー
　～・コードとエラーの起きた行番号を
　　得る …… 文 79
　～発生時処理ルーチンを指定する
　　…… 文 176
　～発生の模擬やエラー・コードの定義を
　　行う …… 文 81
円を描く …… 文 22
音継続の処理ルーチンを指定する
　　…… 文 180
音の発生 …… 文 274
音楽演奏を行う …… 文 209
音楽バッファーの音の数を返す
　　…… 文 214

か 行

カーソル
　～位置やカーソルの表示形態を指定する
　　…… 文 153
　～の行位置を求める …… 文 49
　～の桁位置 …… 文 219
拡張エラー・コードを得る …… 文 84
型宣言を行う …… 文 58
かな漢字変換による漢字のキー入力
　　…… 解 1-6
環境
　～から値を読み取る …… 文 73
　～にパラメーターを設定する
　　…… 文 71
画面
　～を消去する …… 文 30
　～にデータを表示する …… 文 221
　～にデータを表示する<指定された
　　書式で> …… 文 224
　～の文字の属性を返す …… 文 264
　～モードを設定する …… 文 267
キー
　～操作に割り当てられているコード
　　…… 解 D-19
　～割り込みの処理ルーチンを指定する
　　…… 文 178
　～の定義 …… 文 125
　～の割り込み処理を制御する
　　…… 文 130
キーボード
　～からデータを読み取る …… 文 111
　～から 1 文字を読み取る …… 文 108
　～からデータを 1 行読み込む
　　…… 文 144
記憶域
　～を解放する …… 文 26

～イメージのファイルを読み込む
…… 文 7
～イメージのファイルを保管する
…… 文 9
～から 1 バイト読み取る …… 文 208
～内のプログラムや変数を消去する
…… 文 171
～に 1 バイト書き込む …… 文 218
～の未使用空間のバイト数を得る
…… 文 96
機械語サブルーチン
～の開始アドレスを指定する
…… 文 60
～の呼び出し …… 文 294
～の処理手順 …… 解 B-1
～を呼び出す …… 文 10
逆正接を求める …… 文 3
行番号
～を自動的に作成する …… 文 4
～を付け換える …… 文 246
区点コードによる漢字のキー入力
…… 解 1-7
経過時間ごとの割り込み処理を制御する
…… 文 290
ディレクトリーを変更する …… 文 16
コード体系の詳細 …… 解 D-1

さ 行

サブルーチン
～から戻る …… 文 252
～に行き，戻る …… 文 101
～に行き，戻る＜式の値に従って＞
…… 文 177
座標変換を行う＜ワールド座標系とスクリーン座標系間の＞ …… 文 215
出力行の幅を設定する …… 文 310
出力ポートへ 1 バイト送る …… 文 199
処理の繰り返しを設定する …… 文 92
式の値を変数に代入する …… 文 139
指数を得る …… 文 83
自然対数を求める …… 文 158
順次ファイル
～からデータを 1 行読み込む
…… 文 145
～のデータを変数に読み取る
…… 文 114
～の入出力手順 …… 解 A-5
～への書き込みを行う＜区切り文字付き＞ …… 文 318
条件が真のときにループ内の処理を実行する
…… 文 308
実行
～を再開する …… 文 44
～を中断する …… 文 280
～の終了 …… 文 70
時刻の読み取りと設定を行う …… 文 287
数式をストリングに変換する …… 文 281
数値をストリングに変換する …… 文 169
数値精度と有効範囲 …… 解 2-14
ストリング
～を繋げるには …… 解 2-34
～から数値に変換する …… 文 295
～型を数値型に変換する …… 文 50
～内の文字の位置を得る …… 文 134
～内の文字を調べる …… 文 119
～の一部を得る …… 文 166
～の置き換え …… 文 164
～のスペース指定 …… 文 277
～のバイト数を得る …… 文 138
～の左から指定したバイト数分の文字を

得る …… 文 136
～の左寄せと右寄せ …… 文 162
～の右から指定バイト数の文字を返す …… 文 253
～の右寄せ …… 文 259
スピーカーを鳴らす …… 文 6
図形
～を画面に置く …… 文 235
～の読み取り …… 文 99
正弦を求める …… 文 273
正接を求める …… 文 286
整数に変換する …… 文 121，文 21，文 91
セグメント・アドレスを定義する …… 文 57
線
～を描く …… 文 64
～や矩形を描く …… 文 140
絶対値を求める …… 文 1
装置駆動ルーチン
～へ制御データを渡す …… 文 122
～へ渡した制御データの実行を確認する …… 文 123
装置のエラー情報を得る …… 文 77

た 行

単精度に変換する …… 文 47
注釈文を設定する …… 文 245
通信制御信号を操作するには …… 解 C-10
通信ファイル
～の入出力手順 …… 解 C-1
～をオープンする …… 文 192
通信割り込み
～の処理ルーチンを指定する …… 文 174
～処理を制御する …… 文 42
定数を記憶する …… 文 52
点を描く …… 文 220，文 232
点の色を得る …… 文 216
ディスク中のファイル名を表示する …… 文 89
ディレクトリー
～の削除 …… 文 255
～を作成する …… 文 168
データをファイルへ書き込む …… 文 229
データの画面表示を行う＜区切り文字付き＞ …… 文 317
特定の行へ行く＜式の値に従って＞ …… 文 177
トレースの開始と中止を設定する …… 文 291

な 行

入力ポートから 1 バイトを得る …… 文 110

は 行

配列
～を消去する …… 文 76
～宣言を行う …… 文 62
～の添字の最小値を変更する …… 文 198
判断を行う …… 文 105
倍精度に変換する …… 文 11
パレットに色を設定する …… 文 206
表示すると同時に印刷するには

…… 解 1-14
日付の読み取りと設定 …… 文 53
頻繁に使用する予約語を表示させるには
…… 解 1-13
ビューポートの設定を行う …… 文 301
ファイル
～をオープンする …… 文 185
～をクローズする …… 文 28
～を削除する …… 文 132
～から文字を読み取る …… 文 116
～から文字を読み取る …… 文 118
～内の現在の参照位置を得る
…… 文 152
～のクローズとシステム・バッファーの
消去を行う …… 文 248
～の終了状態を返す …… 文 75
～の長さを得る …… 文 157
～名を変更する …… 文 170
符合を調べる …… 文 270
分岐する …… 文 103
プリンター
～出力の設定 …… 文 160
～に出力する …… 文 149
プリント・ヘッドの現在位置を知る
…… 文 159
プログラム
～・リストを表示する …… 文 147
～をマージする …… 文 163
～を記憶域に読み込む …… 文 150
～を実行する …… 文 260
～を保管する …… 文 262
～行を削除する …… 文 61
～行を表示する …… 文 69
～作成から実行までの手順
…… 解 1-19
～の再実行 …… 文 250
平方根を求める …… 文 279
変数
～型を表す文字 …… 解 2-11
～データが記憶域に占める領域
…… 解 2-10
～の値を交換する …… 文 283
～のアドレスをストリングで得る
…… 文 300
～やファイル制御ブロックのアドレスを
得る …… 文 298

ま 行

文字
～を指定個数並べたストリングを
作成する …… 文 282
～数を得る …… 文 133
～表示範囲を定義する …… 文 305
模様を塗る …… 文 200

や 行

ユーザー関数の定義 …… 文 55
余弦を求める …… 文 46

ら 行

乱数を発生させる …… 文 257
乱数シードを再設定する …… 文 241
ランダム・ファイルの入出力手順
…… 解 A-15
ループを閉じる …… 文 172
レコード
～を書き込む …… 文 234
～を読み込む …… 文 97

～の各項目を定義する …… 文 86
連鎖されたプログラムに制御を渡す
…… 文 43
連続コード
～と内部コードを使用した漢字のキー入力 …… 解 1-9
～を得る …… 文 2
～を文字に変換する …… 文 18
論理演算子の働きについて …… 解 2-28

英字

BASIC
～を始動するには …… 解 1-1
～コマンドに指定できるパラメーターについて …… 解 1-29
～のプログラムの形態について
…… 解 1-15
DATA ステートメントで設定された値を読み取る …… 文 243
DATA ステートメントの行指定
…… 文 249
DOS
～に戻る …… 文 284
～のコマンドを実行する …… 文 271
JIS16 進コードを得る …… 文 124
JIS 区点コードを得る …… 文 135
PRINT と共に使用してスペースの設定を行う …… 文 278
PRINT と共に使用して桁そろえを行う
…… 文 285

数字

10 進数を 16 進数に変換する
…… 文 104
10 進数を 8 進数に変換する …… 文 173
1 バイト・コードの英数字を 2 バイト・コードの英数字に …… 文 12
2 バイト・コードの英数字を 1 バイト・コードの英数字に …… 文 48

索引

あ 行

値の受け渡し …… 解 B-10
色 …… 解 3-21，文 32
　～の使用と画面モード …… 解 3-26
印刷 …… 文 160
　～制御文字の指定 …… 解 E-8
　～の指定を行うキー …… 解 1-14
ウインドー …… 文 313
　～とビューポート …… 解 3-16
英数字の入力 …… 解 1-4
エラー …… 文 A-1，文 81，文 84
円 …… 文 22
オープン …… 文 185

か 行

カーソル …… 文 153
　～移動キー …… 解 1-11
　～の行位置 …… 文 49
　～の桁位置 …… 文 219
解放 …… 文 26
拡張コード …… 解 D-17
型 …… 文 58
カタカナの入力 …… 解 1-5
型宣言文字 …… 解 2-11
かな/漢字の通信 …… 解 C-8
かな漢字変換 …… 解 1-6
カラー・システム …… 解 3-11
　～文字モード …… 解 3-27
環境 …… 文 71
漢字の入力 …… 解 1-5
区点コード入力 …… 解 1-7
関数 …… 文 55
間接モード …… 解 1-21
関係演算子 …… 解 2-24
画面
　～イメージ …… 文 99
　～の使用 …… 解 3-8
　～表示 …… 解 3-33
　～モード …… 解 3-9，文 267
キー操作 …… 解 1-3
キーボード・バッファー …… 解 E-7
キーボードからの入力 …… 文 108，文 111，文 116，文 118
キー割り込み …… 文 125，文 130
記憶域マップ …… 解 E-1
機械語サブルーチン …… 解 B-1
　～処理概略 …… 解 B-1
逆正接 …… 文 3
行　～の削除 …… 解 1-26
　～の消去 …… 解 1-24
　～の追加 …… 解 1-24
　～番号 …… 解 1-16
クローズ …… 文 28
グラフィック・モード …… 解 3-14
グラフィックス …… 解 F-1
連続コード …… 解 D-1，文 2，文 18
現行ディレクトリー …… 解 3-7
コード体系 …… 解 D-1
交換回線と専用回線 …… 解 C-8
構文エラー …… 解 1-27
固定小数点数 …… 解 2-7

さ 行

最大作業領域 …… 解 1-31
作業領域 …… 解 E-2
削除 …… 文 61, 文 132, 文 171
サブルーチン …… 文 101
　～を記憶域に置く …… 解 B-2
　～の呼び出し …… 解 B-18
算術演算子 …… 解 2-22
雑音対策 …… 解 C-9
座標指定 …… 解 3-14
終了 …… 文 70, 文 284
出力
　～制御信号 …… 解 C-13
　～幅 …… 文 310
消去 …… 文 30, 文 76
指数 …… 文 83
システム・リセット …… 解 1-2
自然対数 …… 文 158
順次ファイル …… 解 A-5
　～のプログラム例 …… 解 A-10
　～の書き込み …… 解 A-7
　～の読み取り …… 解 A-9
乗算 …… 解 2-22
定数 …… 解 2-7
剰余の計算 …… 解 2-23
実行を制御するキー …… 解 1-13
実行順序 …… 解 2-31
次元 …… 解 2-18
時刻 …… 文 287
数式の演算子 …… 解 2-21
数値
　～定数 …… 解 2-7
　～精度 …… 解 2-13
　～精度の変換 …… 解 2-15
　～の入力 …… 解 1-10
　～比較 …… 解 2-24
　～変数 …… 解 2-10
　～変数の記憶 …… 解 E-4
スクリーン座標系 …… 解 3-16
ステートメント …… 解 1-16
ストリング …… 解 2-6
　～定数 …… 解 2-7
　～の演算子 …… 解 2-33
　～の長さ …… 文 138
　～比較 …… 解 2-25
　～変数 …… 解 2-10
　～変数の記憶 …… 解 E-3
スピーカー …… 文 6
制御信号の操作 …… 解 C-10
制御文字と連続コード …… 解 D-6
正弦 …… 文 273
整数 …… 解 2-7, 文 21, 文 91,
　文 121
整数除算 …… 解 2-22
正接 …… 文 286
セグメント …… 文 57
線 …… 文 64, 文 140
絶対値 …… 文 1
走査コード …… 文 B-8
送信停止と送信開始 …… 解 C-2
装置駆動ルーチン …… 文 122
装置名 …… 解 3-4
添字 …… 解 2-19, 文 198

た 行

縦横比 …… 文 23
単色
　～グラフィック・モード

…… 解 3-28
～システム …… 解 3-10
～文字モード …… 解 3-26
単精度 …… 文 47
単精度数 …… 解 2-14
注釈 …… 解 1-17
直接モード …… 解 1-21
通信 …… 解 C-1
～エラー …… 解 C-15
～端末 …… 解 C-4
～バッファー …… 解 1-31
～ファイルの入出力 …… 解 C-1
定数の指定 …… 文 52
ディレクトリー …… 解 3-6
～情報 …… 文 89
～の構造 …… 解 3-6
～のタイプ …… 解 3-7
特殊文字 …… 解 2-2
トレース …… 文 291
ドライブ指定子 …… 解 3-4

な 行

内部コード …… 解 D-1
入力制御信号 …… 解 C-10

は 行

ハード・ディスク …… 解 1-2
配列 …… 解 2-18, 文 62, 文 76
～の定義 …… 解 2-18
半 2 重と全 2 重 …… 解 C-8
判断 …… 文 105
倍精度 …… 文 11
～数 …… 解 2-14
パス名 …… 解 3-3
パレット …… 解 3-29, 文 206
標準出力指定 …… 解 1-30
標準入力指定 …… 解 1-30
日付 …… 文 53
～と時刻の入力 …… 解 1-1
非同期通信 …… 解 C-1
ひらがなの入力 …… 解 1-5
ビューポート …… 文 301
ファイル …… 解 3-1
～指定 …… 解 3-2
～制御ブロック …… 解 E-5, 文 298
～の終り …… 文 75
～の送受信 …… 解 C-6
～の長さ …… 文 157
～番号 …… 解 3-1
～名 …… 解 3-5
符号 …… 文 270
浮動小数点除算 …… 解 2-22
～数 …… 解 2-8
負の記号 …… 解 2-22
フロア関数 …… 文 121
物理座標系 …… 解 3-16
分岐 …… 文 101, 文 103
プリント・ヘッド …… 文 159
プログラム
～・ファイルの管理 …… 解 A-1
～を編集するキー …… 解 1-11
～を入力する …… 解 1-11
～の作成 …… 解 1-19
～の作成と実行 …… 解 1-19
～の削除 …… 解 1-26
～の修正 …… 解 1-22
～の実行 …… 解 1-19, 解 A-2
～の実行モード …… 解 1-21

～の保管 …… 解 1-19，解 A-1
～の読み込み …… 解 1-19，解 A-2
平方根 …… 文 279
編集キー …… 解 1-12
変数 …… 解 2-10
～型 …… 解 2-11
～のアドレス …… 文 300
～の記憶方法 …… 解 E-3
～の内容を調べる …… 解 1-28
～名 …… 解 2-11
べき乗 …… 解 2-22

ま 行

文字
～を入力するキー …… 解 1-4
～数 …… 文 133
～の位置 …… 文 119，文 134
～の削除 …… 解 1-23
～の挿入 …… 解 1-24
～の変更 …… 解 1-23
～モード …… 解 3-12
モデム装置制御信号 …… 解 C-11
模様 …… 文 200

や 行

余弦 …… 文 46
予約語 …… 解 2-3
～を表示するキー …… 解 1-13

ら 行

乱数 …… 文 257
ランダム・ファイル …… 解 A-15
～のプログラム例 …… 解 A-22
～の書き込み …… 解 A-18
～の読み取り …… 解 A-20
リストの表示 …… 解 1-19
ループ …… 文 92，文 308
連結演算子 …… 解 2-34
連鎖 …… 文 13
連続コード
～と内部コードの関係 …… 解 D-10
～と内部コード入力 …… 解 1-9
論理演算子 …… 解 2-26
～の働き …… 解 2-28
論理座標系 …… 解 3-16

わ 行

ワールド座標系 …… 解 3-16

英 字

AND …… 解 2-27
BASIC
～コマンド …… 解 1-1
～で使用できるデータ …… 解 2-6
～で使用できる文字 …… 解 2-1
～の始動 …… 解 1-1
～の始動時に指定できる項目 …… 解 1-29
～のプログラム …… 解 1-15
～ファイル制御ブロック …… 解 E-5
DOS
～に戻る …… 解 1-1
～プロンプト …… 解 1-1
EQV …… 解 2-27

IMP …… 解 2-27
JIS16 進コード …… 解 D-1,
文 124
〜と内部コードの関係 …… 解 D-13
〜定数 …… 解 2-9
JIS 区点コード …… 解 D-1, 文 135
〜と内部コードの関係 …… 解 D-11
JIS 区点定数 …… 解 2-9
NOT …… 解 2-27
OR …… 解 2-27
PF キー …… 文 125
XOR …… 解 2-27
/C …… 解 1-31
/D …… 解 1-31
/F …… 解 1-30
/I …… 解 1-30
/M …… 解 1-31
/S …… 解 1-31

数字

16 色カラー・グラフィック・モード
…… 解 3-30
16 進数 …… 解 2-8, 文 104
1 バイト・コード …… 文 12, 文 48
〜文字セット …… 解 D-3
2 バイト・コード …… 文 12, 文 48
〜文字セット …… 解 D-9
4 色カラー・グラフィック・モード
…… 解 3-29
8 進数 …… 解 2-9, 文 173

BASIC インタープリター ユーザーズ・ガイド（下巻）文法編
N:SH18-2120-2

IBM®
日本アイ・ビー・エム株式会社
東京都港区六本木３－２－１２　〒106
TEL(03)586-1111（代）

Printed in Japan